TRABAJO DE GRADO

CONSULTAS EN APLICACIONES HIPERMEDIALES

Universidad Nacional de La Plata

Directores : Alicia Diaz , Gustavo Rossi
Alumno : Marcos Dvoskin

Abstract

Hoy en día, el método de acceso primario para acceder a la información que contiene una aplicación hipermedial es la navegación : que significa ir de un nodo del hipermedia al otro a través de los links existentes en el mismo. Esta forma de acceder queda obsoleta si estamos accediendo a un hipermedia grande o extenso, ya que se pierde mucho tiempo intentando buscar la información deseada y a la vez crea el "**gran problema de desorientación**" que sufren los usuarios de aplicaciones hipermediales. Entonces, el objetivo del trabajo de grado es construir un servidor de consultas que resuelva este problema.

Para ello definiremos un lenguaje de consultas que estará basado en el lenguaje SQL del Álgebra Relacional. Las aplicaciones que se conecten al servidor de consultas podrán efectuar una consulta en el lenguaje definido y obtener la respuesta a la misma, evitándose de esta forma tener que perder tiempo intentando encontrar la información deseada y contribuyendo así a resolver el problema de la desorientación.

INDICE

CAPITULO 1

APLICACIONES HIPERMEDIALES

CONTENIDO

En este capítulo se hace una introducción a las aplicaciones hipermediales. Para ello en la primer sección haremos una referencia histórica, para saber como se originó este concepto. En la segunda sección, con el objetivo de analizar como se aplican los conceptos introducidos en la primer sección, describiremos Notecards, que fue uno de los primeros sistemas de hipermedia que se construyeron. En la tercer sección, haremos una crítica a Notecards, que a la vez sirve como una "agenda" de las problemáticas que existen en los sistemas de hipermedia. En la cuarta y última sección hacemos una conclusión basada en lo que considero uno de los problemas más grandes de este concepto : la necesidad de una herramienta que contribuya a eliminar el gran problema de desorientación existente al navegar los grandes sistemas de hipermedia.

Debido a que este trabajo está relacionado con las aplicaciones hipermediales, consideramos importante hacer una buena introducción a las mismas.

1.1 BACKGROUND HISTORICO : COMO EMPEZO TODO

Primero definamos lo que es un hipertexto : forma de administrar información en la cual los datos se almacenan en una red de nodos y links. Los nodos pueden contener texto , gráficos, audio, video e incluso código u otras formas de datos. Los nodos son mostrados a través de un browser interactivo y manipulados a través de un editor.
El término hipertexto fue manipulado por Ted Nilson durante los '60, pero el concepto puede remontarse al año 1945, cuando Vanevar Bush describió el concepto "memex" como lo siguiente :

> " un dispositivo en el cual un individuo puede almacenar sus libros o anotaciones, los cuales pueden ser consultados con gran velocidad y flexibilidad. Este dispositivo sería un suplemento o extensión de la memoria del individuo"

Aclaración: cuando se habla de red nos referimos al grafo que forman los nodos conectados por los links.

El primer intento de construir un memex no tuvo lugar hasta 20 años más tarde de la descripción de Bush. En 1968, Doug Engelbart, en el instituto de investigación de Standford, mostró un documento con las características de un hipertexto. Al mismo tiempo introdujo una gran invención : el mouse.
En los siguientes 20 años a esta demostración de Engelbart, el interés y la actividad sobre hipertextos comenzó a brotar. En [*SMIT*88] se menciona que varios trabajos fueron los precursores de este nuevo concepto:

-ZOG: un sistema de alta perfomance desarrollado en la universidad de Carnegie-Mellon. Zog [*ROBE*81]es el predecesor de KMS (un sistema comercial, en [YODE87] podemos encontrar una descripción del mismo).

- INTERMEDIA: un conjunto de herramientas que permiten al autor de una aplicación unir información representada con texto, gráficos o video sobre una red de work-stations de alta perfomance.

- NOTECARDS : el sistema más ambicioso de los '70, desarrollado en XEROX PARC. Notecards lo describiremos en una sección más adelante, ya que nos dará la base para una discusión acerca de lo que hoy en día necesitan los sistemas de hipertextos.

- NEPTUNO : un sistema de hipertexto para asistir la ingeniería de software, desarrollado en Textronix.

Durante 1987 el interés en los hipertextos creció rápidamente. Ningún concepto aislado pudo explicar esto, fueron varios factores : work-stations poderosas, monitores con alta resolución ,

redes de comunicaciones y los bajos costos de todos estos recursos favorecieron este crecimiento.
Sin embargo, 2 eventos específicos parecieron haber jugado un rol muy importante. El primero de ellos fue la introducción de Hipercard por Apple. Mientras que este sistema es relativamente primitivo comparado con algunos anteriores, la promoción agresiva del mismo hizo que el concepto de hipertexto pase de ser conocido sólo por unos pocas personas a ser conocido por millares que son usuarios de algún sistema de hipertexto. El segundo evento fue Hipertext ' 87, la primer conferencia hecha especialmente para hipertextos. Trajo participantes de 5 continentes. El encuentro incluyó científicos de distintas áreas como administración de la información, tratamiento de texto e imágenes, ingeniería de software, diseño VLSI, gráficos e interacción hombre-máquina. También se hicieron presentes academias de distintas ciencias como Medicina, Filosofia, Sicología e incluso fueron presentados intereses económicos por parte de personas que desarrollan sistemas, vendedores y también grupos que planeaban desarrollar sistemas de hipertextos para áreas específicas. Hubo usuarios, que mostraban las necesidades de usar hipertextos en distintas áreas.
A partir de esta conferencia, que atrajo el interés de personas pertenecientes a una gran variedad de especialidades, empezaron las conferencias, encuentros , publicaciones y demás eventos que contribuyeron al desarrollo de este nuevo concepto.

Para entender porqué el hipertexto atrae tanto la atención, debemos entender cómo un hipertexto difiere de un documento convencional.
En la mayoría de los documentos convencionales, como novelas o artículos, la estructura física y lógica del mismo están muy relacionadas. Físicamente el documento es una secuencia lineal de palabras, que fue dividida en líneas y páginas para conveniencia y comodidad del que la va a leer. Lógicamente el documento es también lineal; las palabras están combinadas para formar oraciones, éstas para formar párrafos y éstos para formar secciones y así sucesivamente. Estos tipos de documentos llevan a los lectores a leer en forma lineal, desde el principio hasta el final, empezando por la introducción, luego la primer sección, después la segunda y así sucesivamente hasta llegar al final.
Sólo unos pocos documentos convencionales, como enciclopedias o diccionarios, separa la estructura física de la lógica. Físicamente, estos documentos son secuencias de unidades independientes, donde cada unidad trata un tema específico. Lógicamente son más complejos. El lector rara vez lee estos documentos desde el principio hasta el final, pero sí ubica la unidad o el tema específico que le interesa (asociando este mecanismo con el acceso directo) y lee esa parte específica del documento en forma secuencial. Sin embargo, el lector se puede encontrar con frases como " también ver " durante esa lectura secuencial. Para seguir estas referencias, el lector debe encontrar el volumen apropiado, luego la unidad y dentro de ésta, la sección de su interés. Acá podemos apreciar que la estructura lógica de referencia es más compleja. Los hipertextos proveen estos mecanismos de referencia y también otras nuevas características.

Anteriormente describimos un hipertexto como un documento en el cual la información es presentada como nodos conectados por links. Cada nodo puede ser pensado o imaginado como una sección de una enciclopedia. Los links unen una sección con otra para formar una unidad, y también unen las unidades para formar la enciclopedia.

Estos links son mostrados en los nodos de una forma especial, para que el lector sepa que se trata de una parte especial del nodo que lleva a otro nodo (esto se lograría haciendo un click sobre esa parte especial). De esta forma el usuario va navegando de un nodo a otro siguiendo estos links hasta encontrar la información deseada. También debe existir un mecanismo para volver al nodo anterior al que se está visualizando en el momento.

Se podría establecer una analogía entre los documentos convencionales y los hipertextos, pero estos últimos son más flexibles. Uno podría leer un hipertexto de la misma forma que lee un documento convencional : primero se lee la introducción, luego los nodos de la primer sección, luego los de la segunda y así sucesivamente hasta el final. Sin embargo, uno puede leer secciones en un orden diferente. Aparte, en los documentos convencionales seguir una referencia significa moverse a lo largo de una docena (o más) de volúmenes pesados, esto es una tarea bastante pesada.
La atracción por los hipertextos crece cuando éstos son implementados en ambientes sobre red o workstations. Mientras que seguir una referencia en una gran enciclopedia puede llevar varios minutos, muchos sistemas de hipertextos pueden mostrar el siguiente nodo en pocos segundos.
Otras desventajas de los documentos convencionales es que están limitados a mostrar información en forma de texto o gráficos, pero los nodos de un hipertexto pueden mostrar lo anterior más sonido, secuencias de video, animación e incluso programas que son ejecutados cuando el nodo donde se encuentra este programa es navegado o visualizado. La estructura secuencial y las referencias de un documento convencional son fijas al momento de imprimir el documento. Pero los nodos y links de un hipertexto pueden ser cambiados en el tiempo. La información en un nodo puede ser actualizada, nodos nuevos pueden ser agregados a la estructura existente y también posibles links para reflejar sus relaciones.
Todas estas ventajas hacen que las personas vayan tomando conciencia de que existe un nueva forma de almacenar y mostrar información.

Antes de continuar con este informe , dejemos en claro ciertos conceptos que se han manipulado a lo largo de esta sección :

HIPERMEDIA: este en realidad es un concepto que lo nombramos implícitamente.
Podemos decir que una aplicación es de hipermedia si cumple las siguientes características:
a) La información está almacenada en un conjunto de nodos multimedia.
b) Los nodos están implícita o explícitamente organizados en 1 o más estructuras (usualmente los nodos que estén relacionados estarán conectados por links).
c) El usuario accede a la información a través de la navegación de estos nodos.

HIPERTEXTO : es el mismo concepto que el anterior , nada más que sus nodos están limitados a mostrar texto (de allí su denominación de HiperTEXTO).

SISTEMAS DE HIPERTEXTO/HIPERMEDIA : son sistemas que sirven para generar un hipertexto/hipermedia. ToolBook es un ejemplo de sistema de hipermedia.

1.2 NOTECARDS : UNO DE LOS PIONEROS EN SIST. DE HIPERMEDIA

En esta sección describiremos brevemente un sistema de hipermedia (Notecards), para tener una idea de como se aplican los conceptos desarrollados en la sección anterior. A la vez en la sección siguiente, estudiaremos cuales son las falencias de este sistema para últimamente enunciar las características que debe poseer un buen sistema de hipermedia.

1.2.1 INTRODUCCION

Notecards provee al usuario la posibilidad de desarrollar una red "semántica" de documentos electrónicos conectados por links tipados. Esta red sirve para representar un conjunto de ideas y sus relaciones. También se lo puede ver como un modo de organizar, almacenar y obtener información. El sistema provee al usuario con herramientas para mostrar, modificar y navegar sobre la red. Además , existen un conjunto de métodos y protocolos para crear programas que manipulan la información.

1.2.2 CUATRO CONSTRUCTORES BASICOS DE NOTACARDS

EL sistema fue desarrollado en función a dos constructores primitivos, las tarjetas y los links. También existen 2 especializaciones de tarjetas : browsers y fileboxes, que ayudan al usuario a organizar redes extensas de tarjetas y links.

Tarjetas: es una representación electrónica de un papel de 3x5 cm. Cada tarjeta puede tener texto, gráficos o una imagen construida sobre un mapa de bits. Cada tarjeta tiene un título y puede ser editada para modificar su contenido. Hay varios tipos de tarjetas, diferenciados (en parte) por la naturaleza de la información que contienen (por ej.: texto o gráfico). Se pueden agregar más tipos de tarjetas, ya sean modificaciones de los tipos existentes o tipos completamente nuevos y distintos a los existentes (por ej.: tarjetas de animación).

Links: los links son usados para conectar 2 tarjetas. Cada link posee un tipo y dirección. El tipo es un label que indica la naturaleza de la relación que representa el link. Los links son ubicados en alguna parte de la tarjeta origen a través de un icono que representa un link, pero apunta a una tarjeta destino como un todo. Clickenado sobre el icono que representa un link, se muestra la tarjeta destino en la pantalla.

Browsers: un browser es una tarjeta que muestra un diagrama con la estructura de una red (o subred) de tarjetas. Cada tarjeta de esta red se representa a través de su título. Los links de la red se representan a través de líneas. Los distintos tipos de líneas significan distintos tipos de links. Cada uno de estos títulos que se muestran en el browser son iconos, que llevan a las tarjetas correspondientes:

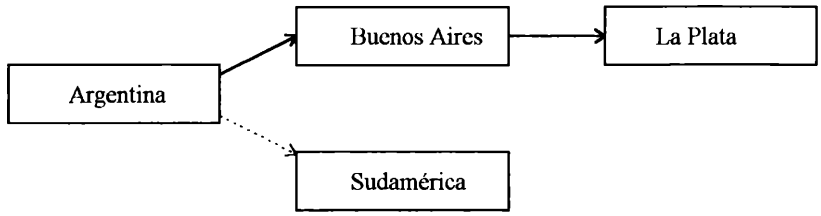


Figura 1

Si clickeamos sobre el título La Plata, se mostrará el nodo que lleva ese título.

FileBoxes: son tarjetas especiales que sirven para organizar o clasificar largas colecciones de tarjetas. Un filebox es una tarjeta en la cual otras tarjetas, incluso fileboxes, pueden ser agrupados. Este tipo de tarjeta fue diseñado para ayudar a los usuarios a organizar grandes redes de datos, motivando a los mismos a crear estructuras jerárquicas para almacenar y obtener información.

1.2.3 INTERACTUANDO EN NOTECARDS

Acceso a la información: la navegación es la forma primaria de acceso. El usuario se mueve en la red siguiendo los links de un tarjeta a la otra. Alternativamente el usuario puede crear un browser sobre una sub-red y moverse desde éste a una de las tarjetas referenciadas en el mismo. Notecards también provee un mecanismo limitado de búsqueda que puede ubicar todas las tarjetas que cumplan con una especificación determinada (por ejemplo: todas las tarjetas que tengan el string "hipertexto" en su título o contenido).

Interface del usuario: la interface del usuario está basada en menúes y el uso del mouse. Las operaciones son iniciadas ya sea por manipulación directa o seleccionando los comandos de los menúes asociados a las ventanas o iconos que se muestran en la pantalla.

1.2.4 ADAPTABILIDAD/INTEGRABILIDAD

Notecards está completamente integrado en el ambiente de programación de Xerox Lisp. Este incluye alrededor de 100 funciones que permiten al usuario crear nuevos tipos de tarjetas, desarrollar programas que controlan o procesan una red, integrar programas Lisp (por ejemplo: un editor de animación) al ambiente de Notecards, y/o integrar Notecards en un ambiente basado en Lisp (por ejemplo: un sistema experto).

Hay un cierto grado de integración o adaptabilidad sin necesidad de programar. El sistema incluye una gran cantidad de parámetros que el usuario puede setear para obtener el comportamiento deseado del sistema (por ejemplo : cómo se deben mostrar los links o el tamaño por defecto de las tarjetas).

1.3 PROBLEMATICAS EN HIPERMEDIA

En [*HALA*87] se destacan siete puntos importantes que nacen de la observación durante unos años del trabajo de los usuarios con Notecards. Estos puntos hacen referencia a Notecards pero a la vez pueden ser tomados como una buena referencia para saber qué características deben poseer los sistemas de hipermedia a desarrollar :

1. Búsquedas y consultas: el método primario para acceder a la información en Notecards es la navegación sobre la red, siguiendo los links de una tarjeta a la otra. También se pueden usar browsers que proveen mapas globales de la red. Estas visiones globales permiten al usuario buscar y moverse directamente a las áreas de la red donde es posible que se encuentre la información que nos interesa.
El acceso navegacional ha sido adecuado y algunas veces ideal para un gran número de aplicaciones. Estas aplicaciones pueden ser divididas en tres clases:
En una **primer** clase el acceso navegacional ha sido suficiente. Aquí encontramos aplicaciones pequeñas relacionadas con tareas de tomar notas y representación informal del conocimiento. En estas tareas, un individuo o grupo de trabajo (de 2 a 3 personas) crea y hace uso intensivo de una red relativamente pequeña (de 50 a 250 tarjetas). Debido a que esta red es chica , los usuarios no tuvieron problemas en encontrar la información que deseaban.

La **segunda** clase de aplicaciones son las que hacen uso intensivo de los browsers, los cuales permiten visualizar la estructura de la red en forma global y esconder los detalles irrelevantes. En esta clase de aplicaciones hay una navegación de browser a tarjeta y de tarjeta a tarjeta.

La **tercer** clase de aplicaciones son aquellas que están relacionadas con las presentaciones interactivas on-line de algún tema o producto. En estas aplicaciones el autor incluye en las tarjetas instrucciones navegacionales para ser seguidas por los lectores.

En contraste a estas aplicaciones, hay una gran variedad en la cual el acceso navegacional es problemático. Las mismas se caracterizan por tener una red muy extensa y heterogénea. En estos casos el usuario se pierde "caminando sin rumbo" sobre la red para poder encontrar cierta información. Los usuarios pueden describir exactamente la información que están buscando, pero simplemente no pueden encontrarla en la red.

Una solución al problema de la navegación es agregar un mecanismo de acceso por consulta. En este caso el usuario puede formular una consulta que encapsule la información que desea buscar y dejar que el sistema la encuentre sobre la red. Notecards provee un mecanismo limitado de búsqueda/consulta (como encontrar las tarjetas que contienen un determinado string de caracteres). Esta búsqueda es extremadamente simple ya que no se puede especificar condiciones booleanas ni expresiones regulares.

Para que Notecards sea útil en el tratamiento de grandes redes de nodos, un mecanismo de consultas debe ser la forma de acceso primaria, junto con la navegación. Proveer un mecanismo de consultas para aplicaciones de hipermedia se ha planteado como un desafío interesante.

Habrían 2 clases de consultas que se necesitarían en las AH. La primera de ellas serían las consultas basadas en contenido. En este caso todas las tarjetas y links son consideradas como entidades independientes y son examinadas individualmente para ver si su contenido coincide con una cierta especificación. Por ejemplo : todas las tarjetas que contienen el string "sistema" . Este tipo de consultas es un poco el standard en varios sistemas de hipermedia.

La búsqueda por contenido ignora la estructura de la red del hipermedia. La segunda clase de consultas son las consultas por estructura. Examinan la red buscando sub_redes que satisfagan un cierto patrón dado. Un ejemplo simple sería : buscar todas las sub_redes de 2 tarjetas conectadas por un link del tipo "soporte" el cual la tarjeta destino sea del tipo "multimedia".

El desarrollo de un mecanismo de consultas es una tarea interesante y dificultosa. Una subtarea sería diseñar el lenguaje de consultas. Este lenguaje lo debería poder usar un usuario típico de hipermedia (sin tener conocimientos especializados de computación). Un segundo desafío es diseñar e implementar un mecanismo para responder consultas hechas en este lenguaje. No es una tarea fácil implementar un mecanismo de búsqueda **eficiente** que actúe sobre la red de nodos y links.

Aparte del rol de buscar información, las consultas tendrán otras utilidades en los sistemas de hipermedia. Por ejemplo, podrían ser usadas como filtro en la interface del hipermedia. El usuario especificará una consulta para describir la información que le interesa ver. Entonces la interface sólo le mostrará aquellas partes del hipermedia que satisfagan la consulta. El browser de Notecards trabaja de esta forma, pero sólo con un muy limitado conjunto de consultas. Es necesario implementar un mecanismo de consultas más potente y flexible.

Búsquedas y consultas van a jugar un papel crucial en las estructuras virtuales. Por lo tanto, el éxito de los sistemas de hipermedia va a depender en gran parte de la solución que den al problema de consultar y buscar sobre la red de nodos y links del hipermedia. En [*MEYR*89] también se detalla porqué se necesita un mecanismo de consultas.

2. Composiciones : extendiendo el modelo básico de nodos y links

Hay sólo 2 constructores primitivos en Notecards, las tarjetas y los links. Todos los otros mecanismos son construidos en función a estos dos.
A pesar de que este modelo ha sido eficiente, la experiencia sugiere que el modelo básico de hipermedia no es suficiente. En particular, la falta de un mecanismo de composición, por ejemplo: una forma de representar y tratar con conjuntos (o sub-redes) de nodos y links como entidades básicas separadas de sus componentes.

Un ejemplo típico para el uso de composiciones en Notecards se puede observar en la tarea de escribir un documento organizado jerárquicamente. En esta tarea el usuario pone el texto de cada sub_sección en una tarjeta. Todas las tarjetas de una sub_sección son agrupadas en un filebox para representar una sección, las cuales se pondrán en otro filebox para representar un capítulo, y éstos en otro filebox para representar el documento. Este esquema funciona. Permite al usuario hacer hincapié en el texto para un capítulo, sección y sub_sección en

particular. Usando el compilador de documentos de Notecards, el usuario puede linealizar la red en una tarjeta documento conteniendo todo el texto/gráfico del documento en el orden apropiado pero sin ninguna estructura jerárquica. Este documento puede ser manipulado (leído o impreso como una unidad simple). Pero hay un problema en el hecho de que la tarjeta documento sea una entidad separada de las tarjetas "fuente" almacenadas en la jerarquía de fileboxes del documento : cambios realizados en el texto/gráfico en la tarjeta documento no son reflejados automáticamente en la correspondiente tarjeta "fuente".
Más aún, el usuario puede ver el documento sólo a un nivel. A pesar de la jerarquía de fileboxes, no hay forma de hacer un zoom/unzoom sobre la estructura del documento para examinar el contenido del documento en diferentes niveles de detalle. Como resultado de esta desventaja muchos usuarios abandonaron Notecards.

La situación planteada anteriormente nos dice que Notecards no tiene un mecanismo para representar composiciones. En particular, colecciones de tarjetas y links vistas como una entidad simple. Si contáramos con un mecanismo, la jerarquía de fileboxes del documento podría ser reemplazada por una jerarquía de composición. En esta última jerarquía, el documento podría ser una composición lineal ordenada de tarjetas capítulo. Cada una de éstas sería una composición lineal ordenada de tarjetas sección, y así sucesivamente hasta tener una composición lineal ordenada de tarjetas "fuente" conteniendo finalmente texto y gráficos.
Cada tarjeta composición en la jerarquía podría ser mostrada con distintos niveles de detalle de sus tarjetas componente. Por ejemplo : la composición de mayor nivel, el documento en sí, podría ser visto como una lista de tarjetas capítulo, o alternativamente, como una concatenación de todos los textos/gráficos que hay en las tarjetas "fuente" de la composición en ese nivel de la jerarquía.

La semántica de las composiciones implica que las componentes de una composición deben ser incluidas "por referencia" y no "por valor" (como en Notecards). Entonces, cambios en las tarjetas fuente deben ser reflejados en las composiciones que tienen esas tarjetas como componentes y viceversa.

En los sistemas de hipermedia, el concepto de composición debe agrupar nodos y links como un concepto primitivo. Notecards, Intermedia y Neptuno no tienen incorporado la noción de composición.
Diseñar un mecanismo de composición para ser incluido en los sistemas de hipermedia trae ciertas preguntas a cuestionarse :

a) ¿Puede un cierto nodo ser incluido en más de una composición?

b) ¿Los links deben referirse entre nodos como una entidad base o se pueden referir a un nodo que existe dentro del contexto de una composición? Si esto último es posible, ¿qué significa navegar este tipo de link?

c) ¿Cómo se pueden manejar versiones de nodos que forman parte de una composición? ¿una nueva versión de un nodo implica ne_

cesariamente una nueva versión de la composición?

d) ¿Las composiciones deben ser implementadas usando links especializados o es un mecanismo totalmente nuevo?

Estas preguntas presentan un desafío para el problema de las composiciones en los sistemas de hipermedia.

3. Estructuras virtuales

NoteCards requiere que sus usuarios segmenten sus ideas en "pedazos" para almacenarlos por separado, uno por tarjeta. Estas tarjetas van a llevar un título y serán agrupadas en al menos un filebox. Estas 3 tareas (aparentemente triviales) les han causado problemas a muchos usuarios. En particular, un usuario que está al principio de la construcción de un documento puede que no tenga bien en claro en este momento el contenido y la estructura del documento. El conocimiento acerca del espacio de ideas, las características que distingue a una idea de la otra y los apropiados nombres para las mismas se irán desarrollando con el transcurso del tiempo. El problema aparece porque la segmentación, titulación y escritura de las tarjetas son requeridas por el sistema al usuario en forma adelantada.
En cierto sentido, Notecards exige a los usuarios una prematura organización de la información. Esto ocurre porque las estructuras conceptuales de los mismos tienden a cambiar más rápido que las correspondientes estructuras en NoteCards. Esto trae la consecuencia de que las últimas son generalmente obsoletas comparadas con el pensamiento actual del usuario. Pero esta situación es inevitable porque siempre va a ser más rápido, fácil y menos tedioso cambiar las estructuras conceptuales que tenemos en nuestro pensamiento que cambiar o actualizar las representaciones externas de las mismas.

La presión hacia una prematura organización también refleja limitaciones en la interface del usuario de Notecards. Los usuarios han pedido que el sistema sea más flexible en este aspecto (por ejemplo : no tener que llenar los títulos de las tarjetas inmediatamente después de crearlas).
Incrementando la facilidad con que las estructuras puedan ser modificadas, haría más fácil a los usuarios llevar la pista en Notecards de las estructuras que internamente en su pensamiento fueron cambiando.

La naturaleza estática de las aplicaciones hipermediales (AH en adelante) podría ser eliminada (cuando convenga) incluyendo al modelo de hipermedia la noción de : estructura virtualmente determinada. En el modelo de Notecards, los nodos y links son definidos por extensión, es decir, especificando la identidad exacta de cada uno de los atributos o componentes de los mismos. Por otro lado, las estructuras virtuales son definidas en forma intensiva, especificando una descripción de sus componentes. Las componentes exactas de una estructura virtual están determinadas por un mecanismo de consulta cada vez que la estructura es instanciada por el mismo. Por ejemplo, una composición virtual de nodos puede ser definida mediante la siguiente especificación : *todos los nodos creados por una persona distinta de mí en los últimos 3 días.*

La noción de estructura virtual para hipermedia es una adaptación directa del concepto de visión (tablas virtuales) en el Álgebra Relacional. En éste, una visión es una tabla construida al momento de aplicar una definición de visión a los datos explícitamente almacenados en las tablas base. Desde el punto de vista del usuario, una tabla basada en una visión es idéntica a una tabla que existe realmente en la base de datos. A pesar de que la noción de visión en un hipermedia podría ser más compleja que en el Álgebra Relacional, el mismo principio de no_diferenciación en la interface (es decir la perspectiva del usuario) se mantiene invariante. Todas las operaciones que se pueden hacer sobre el hipermedia deberían poder hacerse sobre estructuras virtuales.

La noción de estructuras virtuales en hipermedia es posible sólo en un sistema que soporte un mecanismo de consultas sobre la red del hipermedia. La definición de las componentes en estructuras virtuales son de hecho consultas. Instanciar una estructura virtual involucra satisfacer estas consultas y construir entidades dinámicas a partir de las respuestas a estas consultas.

Estructuras virtuales es un mecanismo poderoso cuando se lo combina con la noción de composiciones. Una composición virtual permite al usuario crear nodos que son construidos en el momento del acceso a otros nodos, links y composiciones que están almacenados en la red. Estas composiciones virtuales pasan a ser verdaderas entidades del hipermedia y no simplemente el resultado de una consulta que se muestra por pantalla. Entonces el usuario puede agregar links, propiedades y otras descripciones estáticas a una composición virtual. Los browsers, por ejemplo, pueden ser implementados como composiciones virtuales construidas a partir del resultado a una consulta.

La noción de links virtuales en un hipermedia ha sido explicada en ZOG [*Robe*81] . ZOG incluye un pequeño conjunto de links navegacionales que son construidos cuando un nodo es accedido y mostrado. Estos links conectan al nodo con los nodos visitados recientemente, permitiendo al usuario moverse rápido hacia atrás (de donde vino). Los links virtuales deberían extenderse también a situaciones no navegacionales : en vez de ir a un nodo específico, el usuario tendrá la posibilidad de ir, por ejemplo, *"al más reciente nodo creado que contenga el string hipertexto"* .

Implementar nodos, links y composiciones virtuales no es algo sencillo, especialmente cuando el tiempo es un factor importante. Las estructuras virtuales proveen a los sistemas con la habilidad de adaptarse a la información cambiante en una forma que no sería posible con el modelo estático de hipermedia. A pesar de que las estructuras virtuales no van a reemplazar a las estructuras estáticas (porque no todas las relaciones pueden ser descriptas con una consulta), son una componente crítica de las aplicaciones hipermediales.

4. Mecanismos de computabilidad sobre la red del hipermedia

Notecards es básicamente un sistema pasivo para almacenar y obtener información. Provee a los usuarios con herramientas para definir, almacenar y administrar el hipermedia. En servicio de este objetivo, hace cierto proceso sobre la estructura del hipermedia y la información que

contiene. No hay un mecanismo computacional integrado en Notecards. Lo habría si por ejemplo tuviéramos un mecanismo para resolver consultas estructurales. El hecho de que exista un mecanismo de computabilidad o no en un hipermedia depende de cada aplicación en sí.

5. Versionado

Notecards no tiene un mecanismo de versionado. Cada tarjeta y link está representada y almacenada una sola vez, una sola versión. Como resultado, el rango de aplicaciones que pueden ser soportadas o mantenidas está muy limitado.
Por ejemplo, NoteCards nunca fue usado para mantener software, debido en gran parte, al no tener un mecanismo de versionado.

Proveer un mecanismo de versionado para todas las entidades de un hipermedia trae ciertas dificultades relacionadas con las referencias entre las entidades. En particular una referencia a una entidad podría significar una referencia a una versión específica de la entidad, la versión más nueva de la entidad u otras situaciones análogas. Entonces, cuales de estos tipos de referencia son soportados es una decisión que afecta a todo el hipermedia. Además, las composiciones traen el problema relacionado a la propagación de cambios en las componentes de la composición. Por ejemplo, un cambio en un módulo de software implica la creación de una nueva versión del sistema que contiene el módulo. En contrapartida, modificar la escritura de unas pocas palabras de un párrafo puede que no requiera la creación de una nueva versión del documento que contiene el párrafo.

Mejorar los mecanismos o niveles de versionado es un desafío para la creación de sistemas de hipermedia más flexibles.

6. Soporte para trabajo en grupo

En su diseño original, Notecards hizo hincapié casi exclusivamente en el trabajo individual de personas que trabajan en forma aislada en el procesamiento de ideas. El trabajo en grupo parecía ocurrir fuera de Notecards, a través de conversaciones personales, por correo electrónico u otras formas de comunicación. Pero en la práctica, este no ha sido el caso. La mayoría del procesamiento de ideas se realiza en forma cooperativa o en grupo de personas variando de 2 a 10 trabajando en el mismo proyecto. Por lo tanto debemos decir que es importante que un sistema de hipermedia soporte el trabajo cooperativo a través de mecanismos que permitan, entre otras cosas, el mantenimiento automático de cambios realizados por distintos grupos de trabajo.

7. Extensibilidad/Adaptabilidad

Notecards fue diseñado para ser un sistema extensible. Las expectativas fueron que los usuarios pudieran extender la funcionalidad básica de Notecards para obtener un sistema con mejores prestaciones que se adapte a sus necesidades. El mejor mecanismo para alcanzar este objetivo fue la interface del programador. Prácticamente, la interface del programador fue bastante exitosa. Un gran número de sistemas fueron construidos sobre Notecards. Este fue diseñado para que pueda ser extendido por parte de usuarios que no son programadores

expertos como también por parte de los que lo son. La forma de intentar conseguir esto fue desarrollar mecanismos para la extensión con la característica de que pequeños cambios al sistema podrían hacerse con una pequeña cantidad de trabajo por parte de los usuarios, sin tener un conocimiento extenso de programación y la implementación del sistema. Pero este objetivo no fue alcanzado. Extender Notecards constituye una tarea de programación no trivial que requiere de cierto grado de conocimientos de programación.

El desafío es entonces como hacer para que las aplicaciones hipermediales puedan ser extendidas en un tiempo razonable por usuarios que no son expertos en programación.

1.4 CONCLUSION

El modelo actual de nodos y links no es suficientemente amplio , flexible y completo como para representar y administrar la información que las AH poseen. Si analizamos los ítems mencionados anteriormente, podemos ver que el número 1)Búsquedas y consultas es uno de los más importantes , el cual está relacionado con los ítems 2) y 3), composición y estructuras virtuales respectivamente.

Entonces, el propósito del trabajo de grado es crear un servidor de consultas (basado en un modelo mayor al convencional) para aplicaciones hipermediales y posibilitar que éstas tengan acceso al mismo. De esta forma, cuando el usuario necesite cierta información de la aplicación, deberá indicar al servidor la consulta que quiere hacer, ahorrándose así todo el tiempo de búsqueda que se empleaba en recorrer los nodos de la aplicación hasta encontrar la información deseada (si la encuentra).
Para lograr esto, en el capítulo 3 mostraremos un modelo para especificar las aplicaciones hipermediales sobre las cuales se harán las consultas, introduciendo también un lenguaje de consultas. En el capítulo 4 extenderemos este modelo con otros conceptos para dotarlo de una mayor flexibilidad. Al final de éste haremos un resumen de las características que tiene el modelo y como éste sirve para resolver el problema de búsquedas y consultas visto en el capítulo 1. En el capítulo 5 se describe la implementación del servidor de consultas, en el 6 tenemos ejemplos y clasificación de las AH, y por último en el capítulo 7 el trabajo futuro relacionado con este modelo.

CAPITULO 2

MODELOS EXISTENTES PARA DISEÑAR/CONSULTAR APLICACIONES HIPERMEDIALES

CONTENIDO

En este capítulo contamos con tres secciones. En la primer sección tenemos una introducción a los modelos de diseño/consulta de aplicaciones hipermediales. En la segunda sección describimos Hydesign, que es un modelo orientado a objetos para diseñar aplicaciones hipermediales. La característica principal de este modelo es la riqueza de conceptos existentes en el mismo al momento de diseñar una aplicación. En la tercer sección describimos O2SQL, un lenguaje para realizar consultas sobre una especificación hecha en el lenguaje 02. La característica principal de este lenguaje es que inicia un nuevo estilo de consultas SQL-like. Por último, en la cuarta sección hacemos una conclusión donde comparamos estos 2 modelos.

2.1 INTRODUCCION

De la misma forma que se hicieron necesarias las herramientas de análisis y diseño para la producción de software convencional, también tenemos herramientas o modelos que ayudan al diseñador al momento de construir una aplicación hipermedial, entre ellos podemos mencionar HDM[*SCHW*91] y dos modelos orientados a objetos [*SCHW*94] y [*NANA*91].
Con respecto a los modelos y/o lenguajes para realizar consultas podemos decir que se hicieron varios intentos, los más importantes apuntan a definir un lenguaje de consultas SQL-like (dado que es un lenguaje ya conocido desde el Álgebra relacional y ha podido ser adaptado para hacer consultas sobre aplicaciones de hipermedia).
A continuación presentamos dos modelos de diseño, los cuales a su vez proponen un mecanismo de consultas (que es uno de los problemas más importantes de las AH) que actúa sobre aplicaciones hipermediales descriptas o diseñadas siguiendo el modelo.

2.2 HYDESIGN : UN MODELO ORIENTADO A OBJETOS

Los sistemas de hipermedia se convirtieron en una clase importante de sistemas de información para un rango de nuevos y distintos tipos de aplicaciones. Pero muchos sistemas tienen todavía restricciones. Sólo unos pocos soportan el diseño de objetos o entidades de hipermedia de alto nivel (más allá del modelo básico de nodos y links). Hay restricciones concernientes a la modularización de todo el diseño y el reuso de los recursos en un hipermedia. Hydesign es el prototipo de un sistema de hipermedia extensible que hace hincapié en estas restricciones .

Dos constructores básicos de este modelo son los **links de agregación**, que son el camino para modelar links de distintos tipos, y los **nodos SBL**, que sirven para representar nodos composición (o composiciones) ofreciendo la posibilidad de definir contextos locales con un comportamiento en particular.

La simplicidad conceptual del modelo típico de nodos y links tiene problemas que son bastante conocidos (que los mencionamos en el capítulo anterior). Sin mecanismos adicionales para la estructuración de entidades, los hipermedias extensos o grandes son difíciles de entender y usar por los lectores. Los usuarios muy a menudo se desorientan buscando información de su interés. Para este problema conocido como "el problema de la desorientación" muchas soluciones fueron propuestas. Por ejemplo: fueron introducidos browsers gráficos, que representan subredes de la red global del hipermedia. Pero éstos quedan obsoletos cuando estamos tratando con grandes redes de nodos y links.

Algunos investigadores intentaron resolver el problema de la desorientación de una forma distinta. Se concentraron en extender el modelo de hipermedia con mecanismos de estructuración más flexibles. La idea básica es considerar la posibilidad de tener conjuntos de nodos y links a un nivel más alto que las entidades básicas del hipermedia. También definiríamos operaciones que trabajen sobre estos conjuntos.
Hydesign [*MARM*92] sigue esta línea e introduce varios conceptos que sirven para modelar hipermedias complejos. Se hace distinción entre distintos tipos de links de agregación

(*agregate links*) y nodos de composición de distintos tipos. También la compartición de los recursos del hipermedia es parte del modelo.

2.2.1 CARACTERISTICAS DEL MODELO

Un modelo de hipermedia puede ser entendido como una herramienta formal para describir propiedades estáticas y dinámicas de un hipermedia. Las propiedades estáticas tienen que ver con los aspectos estructurales de las entidades del hipermedia, mientras que las propiedades dinámicas se refieren a las operaciones de creación, manipulación, acceso y eliminación de las mismas. El objetivo de este informe no es mostrar la formalización de estos conceptos pero sí dejar bien en claro qué son y la función que cumplen estos conceptos al momento de diseñar un hipermedia.
Siguiendo un estilo orientado a objetos, podemos ver en la siguiente figura la jerarquía de objetos del modelo Hydesign :

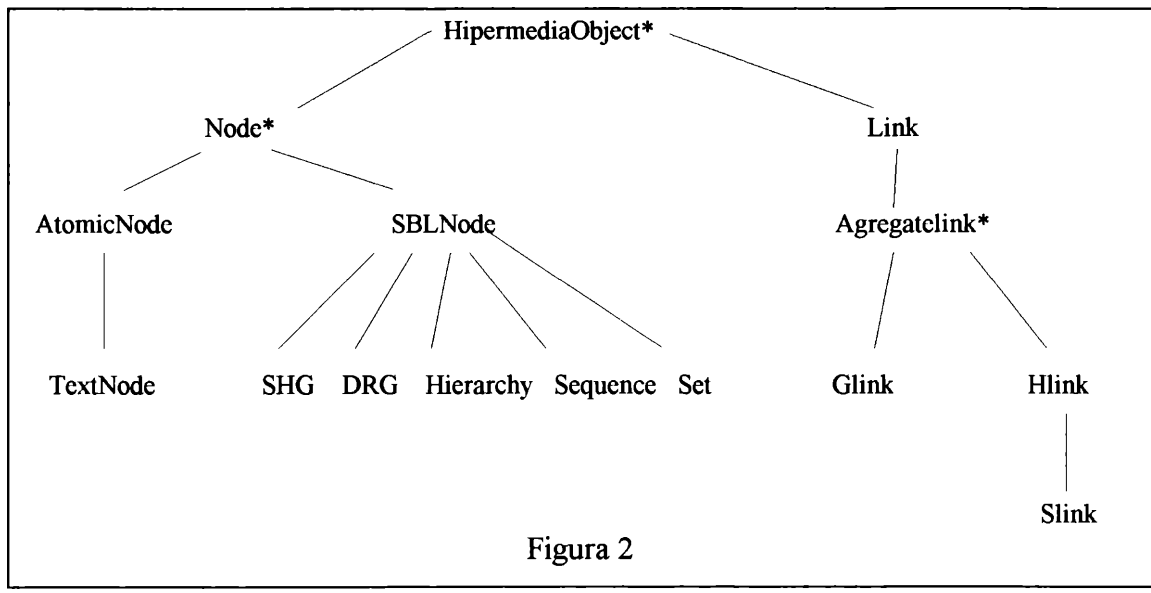


Figura 2

Sólo las clases que no están marcadas con un asterisco, son clases que se pueden instanciar. Las que están marcadas con un asterisco han sido definidas para determinar propiedades estáticas y dinámicas que tendrán todas las instancias de sus subclases (que sean instanciables). En Smalltalk-80, las clases que actúan de esta forma se llaman "superclases abstractas". Si por ejemplo hablamos de instancias de la clase Node, nos referimos a las instancias de las subclases no_abstractas de la clase Node.

En la raíz de la jerarquía, la clase HipermediaObject representa el concepto más abstracto en Hydesign. Acá, son determinadas las propiedades generales para cada objeto de Hydesign, por ejemplo, los atributos *name,category,creator* y *creationDate*. Estos atributos son válidos para cada instancia que se crea. También, es posible agregar un atributo a cualquier clase de la jerarquía de Hydesign. La clase HipermediaObject también define algunas propiedades dinámicas (métodos u operaciones), por ejemplo, operaciones de manipulación para los atributos de los objetos.

A continuación describiremos las propiedades fundamentales de los nodos atómicos, links y nodos SBL, representados por las clases AtomicNode, Link y SBLNode con sus respectivas subclases.

AtomicNode : la característica principal de los nodos atómicos es que tienen (además de los atributos heredados) un atributo adicional llamado *content*, que tiene la información que se quiere mostrar en el nodo. Este atributo no puede ser descompuesto a nivel del modelo, pero desde el punto de vista de una aplicación este atributo puede contener cualquier objeto compuesto.

La clase AtomicNode provee la base para la definición de varios tipos de información (textual, gráfica, auditiva, etc.) que quiere representar una aplicación. TextNode en la figura 2 representa un ejemplo de una aplicación que agregó un tipo específico de nodo.

Como la subclase AtomicNode hereda la funcionalidad de la jerarquía, los nodos pueden ser creados, referenciados (esto se explica más abajo) , copiados, actualizados y eliminados. Sólo las instancias de la clase Node pueden ser el nodo origen o destino de un link. La eliminación de un nodo causa el borrado de los links que salen y llegan al mismo.

Una característica especial de Hydesign es una operación para compartir el atributo *content* de un nodo atómico. Esto es alcanzado mediante la operación *CreateRefNode,* que se basa en la estricta separación de lo que es la funcionalidad de un nodo y el contenido del mismo. Esta operación, aplicada a un nodo existente (llamado nodo original), crea un nuevo nodo (llamado nodo referencia) que se refiere al mismo atributo *content* del nodo original. Por lo tanto, la operación *CreateRefNode* permite usar la misma información en diferentes situaciones representadas por los distintos links de contexto. Un link de contexto para un nodo particular está dado por los links que llegan y salen del mismo. Para ilustrar la idea de este método podemos ver la figura 3. En ella se muestra una porción de la red de nodos y links que representa un curso hipermedia para enseñar bases de datos relacionales.

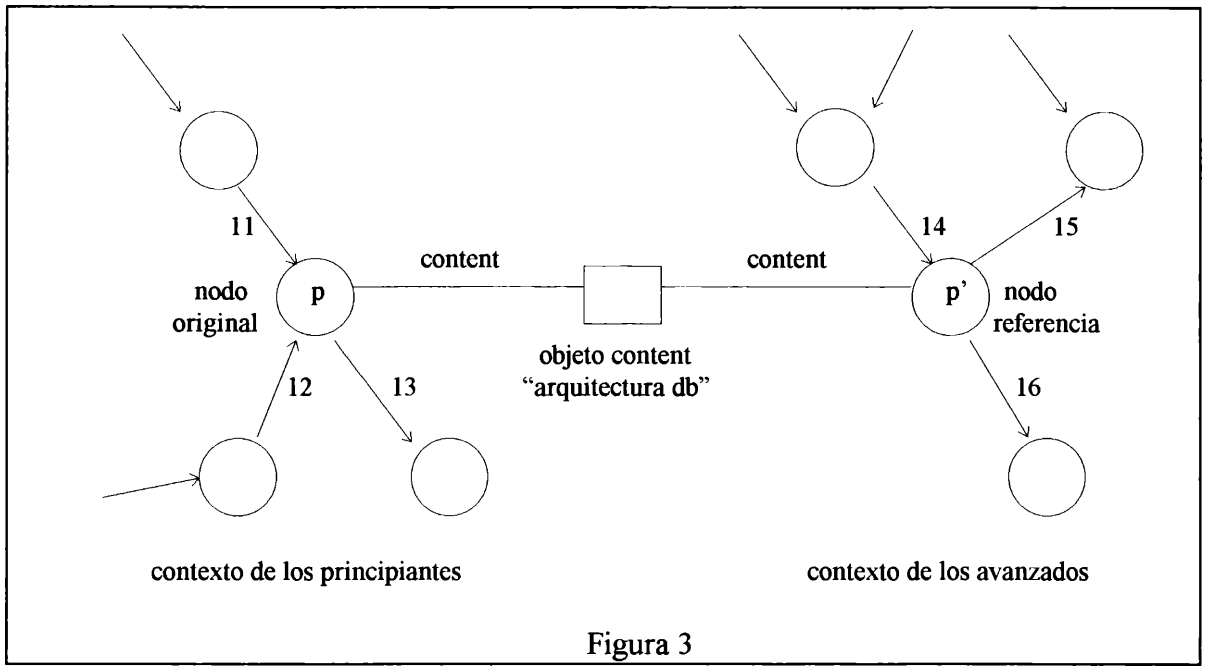


Figura 3

El equipo de diseño creó el hipermedia para ser aprendido tanto por principiantes como por avanzados en el tema. El contenido del atributo *content* es una representación gráfica de la arquitectura de una base de datos y puede ser usado en distintos contextos o situaciones. En el ejemplo, el contenido es tanto útil para el contexto de los principiantes (dado por los links 11,12,13) como para el contexto de los avanzados (dado por links 14,15,16).
El nodo con label p es el nodo original, y el p´ es el nodo referencia que fue creado con la operación *CreateRefNode* hecha al nodo p. Ahora p y p' se refieren al mismo contenido (ya que comparten el atributo *content*). Es claro que cualquier modificación a este atributo será visible para todos los nodos que comparten ese objeto. Los nodos originales pueden ser referenciados más de una vez. Notar que los links con label content no representan los links normales de un hipermedia. Simplemente son referencias a un objeto (como las referencias en los sistemas orientados a objetos) o el concepto de puntero en los lenguajes de programación.

Como Hydesign puede distinguir entre nodos originales y nodos referencia, pueden ser definidas operaciones distintas sobre estos 2 tipos de nodos. Por ejemplo, la modificación del atributo *content* podría ser definida sólo para los nodos originales. Estas componentes reusables y modificables podrían ser agrupadas en lugares especiales (ejemplo : nodos composición) representando nodos librería o nodos archivo. De esta forma la centralización de las operaciones de modificación puede ser alcanzada. Al introducir diferentes operaciones para nodos originales y nodos referencia, tenemos que estos tipos de nodos pueden ser administrados como conceptos distintos también a nivel de presentación de los mismos.

La **eliminación** de nodos originales y referencia está basada en el concepto de **integridad de referencia** ya conocido e implementado en varios sistemas orientados a objetos. Es decir, cualquier nodo (original o referencia) puede ser eliminado sin afectar a otro nodo que comparte el mismo objeto *content*. Esto es porque este atributo o componente no va a ser removido por

el sistema mientras haya una referencia al mismo. También se podrían definir otros **tipos de eliminación** al querer borrar un nodo original:

a- borrar el nodo original y todos los nodos referencia que comparten el atributo *content.*

b- borrar el nodo original incluyendo el objeto *content.* Por lo tanto, los nodos referencia de este nodo original apuntan a algo que dejó de existir (se les puede mandar un mensaje para informarles de la situación), pero el resto del hipermedia queda intacto.

c- borrar el nodo original sólo si no hay nodos referencia creados a partir del mismo.

Links: los links de Hydesign son dirigidos y representan objetos autónomos en el sentido del paradigma orientado a objetos. Diferentes tipos de links son modelados con diferentes clases de links. Los links pueden ser definidos entre instancias de la clase Node. Por ejemplo : entre nodos atómicos y nodos composición (que son los nodos SBL , descriptos más adelante).

La clase Link representa el link normal de referencia. La funcionalidad standard sobre un link incluye la creación, eliminación, navegación y algunas veces operaciones sobre sus atributos. La eliminación de un link referencial no afecta a otro nodo en el hipermedia.
En la subjerarquía, empezando por la clase **AgregateLink**, tenemos otros tipos de links. El propósito de definir estos links de agregación es la de crear objetos o entidades de hipermedia que estén a un nivel más alto que las entidades normales, sobre las cuales se podrán hacer operaciones de borrado, copiado ,consultas, etc. Para alcanzar esto la estrategia básica es redefinir las operaciones standard para la creación y eliminación de links para que tengan incluido el conocimiento acerca de estas entidades de hipermedia más abstractas (por ejemplo: agregar reglas particulares de construcción cuando se crean estos tipos de links). Instancias de la clase AgregateLink son llamadas links de agregación, y una red que ha sido creada con links de agregación se llama agregado de hipermedia *(hipermedia aggregate).*

En Hydesign hay **3 tipos de links de agregación :** la clase **Glink, Hlink** y **Slink.**
Los hipermedias construidos con estos tipos de links tienen una característica en común : representan grafos dirigidos que tienen un nodo raíz.

La clase **Glink** es la más general. Se pueden construir grafos (ver la figura 4) con esta característica a partir de los g-links (instancias de la clase Glink). Durante la creación de un g-link, Hydesign examina básicamente 3 condiciones :

1) ¿ Son el nodo origen y destino del tipo apropiado ?

2) ¿ Hay algún conflicto de links relacionado con otros links de agregación ? (ver coexistencia al final de esta sección).

3) ¿ Es la resultante red un grafo con raíz ?

Si alguna de estas condiciones no se cumple, la creación del link no se efectúa.
Un nuevo agregado de hipermedia es creado si un link de agregación es definido entre 2 nodos que no pertenezcan a otro agregado de hipermedia. El nodo origen del link de agregación pasa a ser raíz.
Si un g-link es eliminado, entonces el sistema debe asegurar que cada nodo y link que ya no se pueden alcanzar más, también sean eliminados. La figura 4 muestra el efecto de la operación de eliminación en un grafo dirigido con raíz construido con g-links. Esta operación de eliminación de un nivel más alto puede ser usada para eliminar fácilmente caminos o subredes.
Los nodos negros son los nodos raíz.

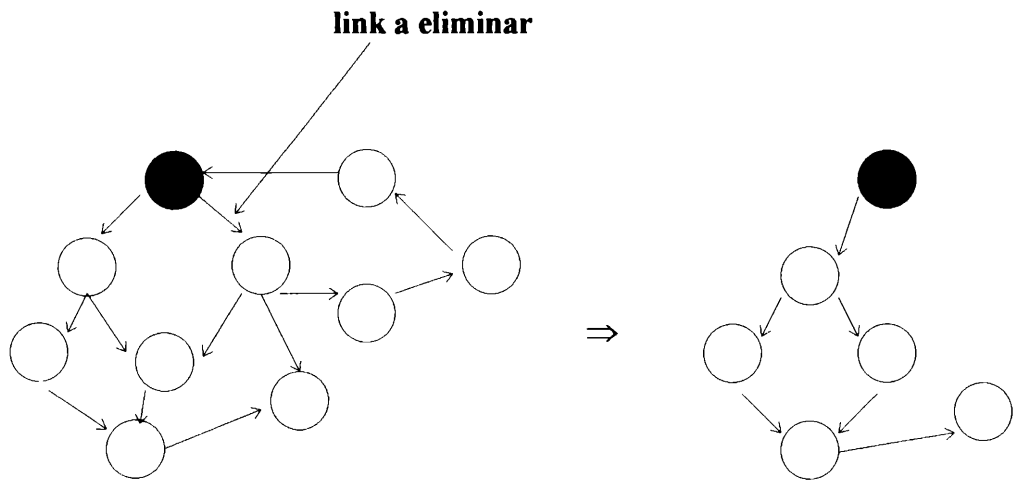


Figura 4

Una eliminación virtual (o reversible) puede ser utilizada para crear configuraciones completamente distintas de la misma red mediante la eliminación virtual de uno o más g-links. De esta forma, se pueden definir **visiones** de la red del hipermedia, las cuales han sido reconocidas como un gran requerimiento de los sistemas de hipermedia. La operación de eliminación virtual de un g-link (como también para otros tipos de links) se llama desactivación. Su complemento, la operación de reactivación, cancela el efecto de la operación de desactivación.

En forma similar, un mecanismo de activación/desactivación ha sido definido para los nodos. Como una visión es un concepto definido por el usuario, el sistema debe ser capaz de administrar distintas configuraciones de la misma red del hipermedia (o partes de el). Este concepto afecta al grafo en su totalidad y cambia el estado de visibilidad de varios objetos del hipermedia.

En forma análoga a los g-links, se han definido los tipos **Hlink** y **Slink**.
Instancias de la clase **Hlink** (que las llamaremos h-links y significa links jerárquicos) tienen una funcionalidad muy parecida a los g-links. La diferencia fundamental es que los h-links constituyen jerarquías, para ser más precisos, árboles dirigidos con raíz en vez de grafos

dirigidos con raíz. La operación de eliminación para un h-link remueve el link y el sub-árbol apuntado por el mismo. En este caso no es necesario checkear condiciones de alcansabilidad como en el caso de los g-links. Desde este punto de vista los h-links representan la clase más fuerte de "dependencia de existencia" entre los nodos que conectan, ya que : un link referencial no tiene impacto sobre el nodo destino cuando se quiere eliminar el link; el nodo destino referenciado por un g-link es eliminado si y sólo si no hay otro camino desde el nodo raíz al nodo destino (excluyendo el g-link a borrar) y el nodo destino referenciado por un h-link es **definitivamente borrado**.

Los link secuenciales (o s-links) son instancias de la clase **Slink,** que es subclase de Hlink. La diferencia fundamental entre los h-links y los s-links es que los últimos forman secuencias en vez de jerarquías. Pueden ser usados por ejemplo, para definir caminos de entidades que se tratan como una unidad simple.

Hay una gran **restricción en Hydesign** con respecto a la coexistencia de distintos tipos de links:
ningún nodo puede formar parte de 2 distintos agregados de hipermedia. Por ejemplo : un nodo no puede ser parte de una secuencia (construida por los s-links) y también parte de una jerarquía (construida por los h-links). Esto incluye también diferentes agregados de hipermedia creados por links del mismo tipo.

Debido al comportamiento simple de los links referenciales (por ej.: operación de borrado), su definición no causa ningún problema. Si no está prohibido por otros mecanismos en Hydesign, los links referenciales pueden definirse siempre. Entonces, es posible crear una conexión navegacional entre 2 agregados de hipermedia en Hydesign.

Nodos SBL: son instancias de la clase SBLNode , representan una extensión al concepto de composición definida por [*HALA*87]. Los nodos SBL son nodos de alto nivel con todas las características de un nodo, pero también con características adicionales indicadas por la abreviatura SBL, que significa **estructura**, **comportamiento** y **localidad**.

Estructura: determina la forma en que los nodos y links pueden ser conectados en un nodo SBL de un determinado tipo.

Comportamiento : determina como van a responder los nodos y links ante ciertas operaciones.

Localidad: los nodos SBL representan ambientes locales que tienen una subred del hipermedia y son independientes de otros ambientes locales. Este concepto es fundamental para ayudar a la modularización de hipermedias grandes y complejos. Los nodos SBL pueden contener nodos atómicos, SBL y links. Por lo tanto, se pueden modelar hipermedias anidados, que permiten al diseñador desarrollar aplicaciones con diferentes niveles de abstracción.

Una forma de construir diferentes tipos de nodos SBL es definirlos sobre la jerarquía de objetos del hipermedia. Esto puede ser realizado, por ejemplo, asociando tipos de links y tipos

de nodos con la definición de una nueva subclase de SBLNode. Las diferentes clases de nodos y links con su significado específico determina aspectos básicos estructurales y de comportamiento de un tipo de nodo SBL en particular.
La clase SBLNode representa el tipo más general de los nodos SBL, una instancia de esta clase puede contener cualquier objeto de la jerarquía de Hydesign. El nivel en el cual un nodo SBL se comporta como nodo atómico se llama nivel externo y el nivel conteniendo la subred del hipermedia se llama nivel interno de un nodo SBL.

En Hydesign, cada objeto, excepto 2 casos, está contenido en exactamente 1 nodo SBL. Con esto, la ubicación de un nodo atómico, SBL o link dentro de la red puede ser identificado unívocamente. La primer excepción es el nodo SBL más externo, que no es contenido en ningún otro nodo SBL. La segunda excepción está dada por una cierta clase de links referenciales, llamados links globales. La característica principal de éstos es que los nodos origen y destino de estos links están en distintos nodos SBL. Por lo tanto, los links globales pueden ser considerados como los "goto" de Hydesign. Desde el punto de vista del usuario los links globales se comportan como cualquier link referencial. Por otro lado tenemos los links locales, cuyos nodos origen y destino se encuentran dentro del mismo nodo SBL. (La figura 6 muestra un ejemplo de un link global que va de un nodo SBL a otro. Cualquier otro link de la figura es local).

Hydesign también da soporte para compartir las principales partes de un hipermedia a través de mecanismos de referencia para los nodos SBL. Pero en comparación con los nodos atómicos, los nodos SBL requieren un trabajo adicional. Aplicando la operación CreateRefSBL a un nodo SBL existente, una nueva instancia de la respectiva clase es creada. El nuevo nodo es llamado nodo referencia, como en los nodos atómicos. Pero debemos mirar el nivel interno de este nodo SBL referencia : se propone un mecanismo de referencia recursivo para tratar su estructura o nivel interno (ver figura 5). Básicamente la operación CreateRefSBL crea para cada nodo del nivel interno un nodo referencia (indicado con la notación ref() en la figura 5). Para cada link se crea una instancia del mismo tipo del link y se copian los atributos desde el link original (indicado con la notación copy() en la figura 5). Es obvio que los valores que representan el nodo origen y destino del link son reemplazados por los nuevos nodos referencia creados. Si el nodo SBL original contiene a su vez nodos SBL, entonces estos nodos también son referenciados. Los nodos referencia que existan en el nivel interno del nodo original también son referenciados. El resultado sigue siendo aún, un nodo referencia.
Este concepto, permite al diseñador extender o manipular el nodo SBL resultante, por ejemplo, con la definición o eliminación de nodos y links sin afectar el nodo SBL original. Los nodos negros son los nodos raíz.

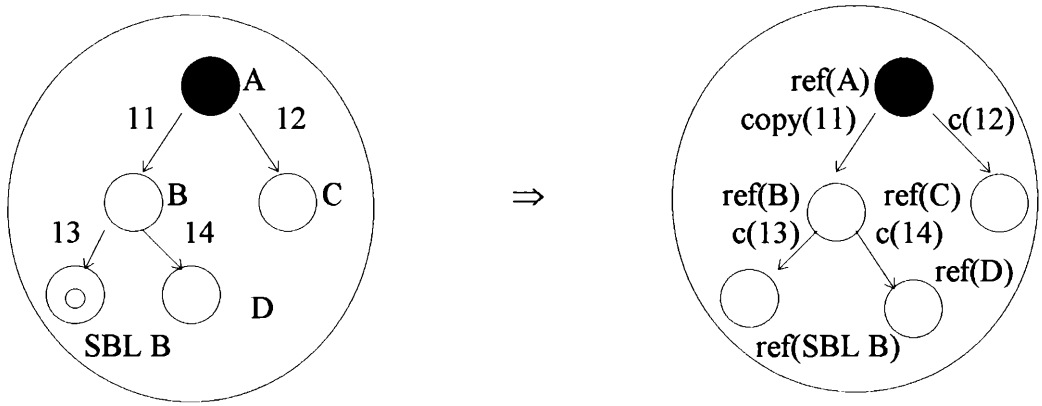
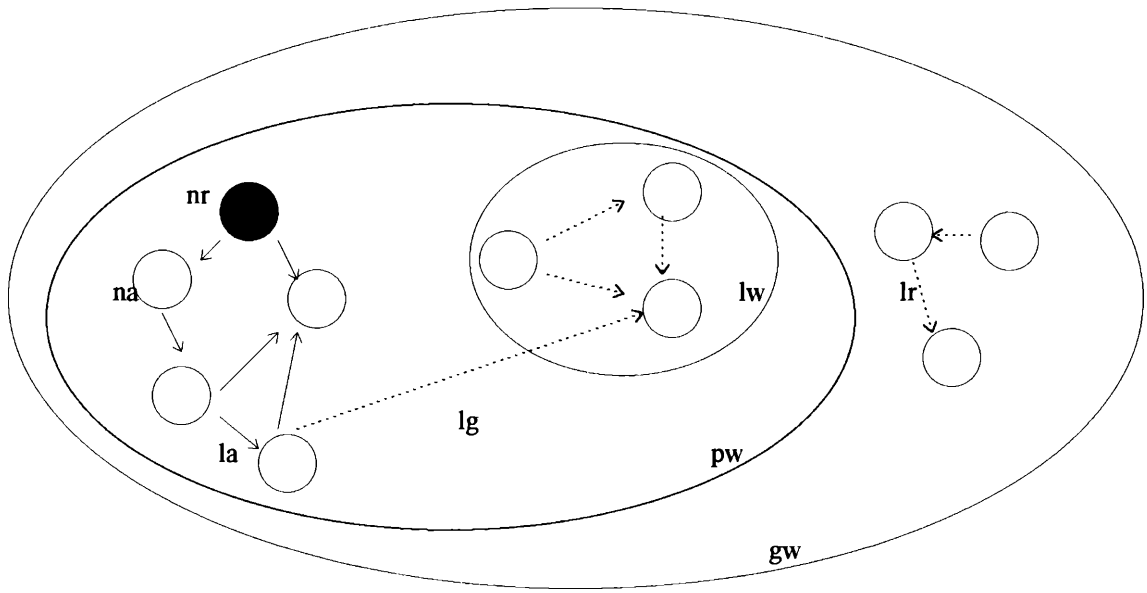


Figura 5

Nota : por cuestiones de espacio escribimos c(12) en vez de copy(12); idem para los otros links

Dos operaciones interesantes sobre nodos SBL son **emptyLinkSet** y **createSBLForm**.
emptyLinkSet borra todos los links dentro de un nodo SBL (esta operación no es recursiva) pero sin usar la operación de eliminación específica para cada tipo de link. Como resultado tenemos un nodo SBL con su conjunto de nodos intacto.
La operación **createSBLForm,** es distinta de createRefSBL en sólo 1 aspecto : cada nodo creado tiene el atributo content vacío. Por lo tanto, los nodos creados no son nodos referencia pero sí son nodos originales en término del modelo de datos de Hydesign. De esta manera el diseñador puede crear un hipermedia que tiene la misma estructura que otro que ya existe. El nuevo hipermedia tendrá sus nodos vacíos (el atributo content no tendrá ningún valor).

La clase SBLnode representa el tipo más general. La subclase **SHG** (standard hipermedia graph) contiene, aparte de nodos de cualquier tipo, sólo links referenciales. Si restringimos los tipos de nodos a que sean atómicos, los nodos SBL de este tipo podrían tener redes del tipo de nodo más común.
La clase **DRG** (directed rooted graph) restringe el conjunto de tipos de links permitidos a los Glink y Link. **Hierarchy** y **Sequence** son especializados de una forma similar.
Los nodos SBL del tipo **Set** no contienen links.



Referencias:
nr : nodo raíz para un agregado construido por g-links
na : nodo atómico
la : link de agregación
lr : link referencial
lg : link global
lw: espacio local. Por ejemplo : un nodo SBL del tipo SHG
pw : espacio privado. Por ejemplo : un nodo SBL del tipo DRG
gw : espacio global. Por ejemplo : un nodo SBL del tipo SHG

Figura 6

La figura 6 muestra un ejemplo con los distintos conceptos desarrollados anteriormente.

2.2.2 CONSULTAS EN HYDESIGN

La mayoría de los sistemas de hipermedia sólo soportan facilidades de consultas simples (como por ejemplo búsqueda de un determinado string en un texto). Un lenguaje de consultas para hipermedia, similar al SQL para bases de datos relacionales, es todavía algo excepcional.

Hydesign provee un lenguaje de consultas SQL-like. Se asume que las consultas se ejecutan sobre el nodo SBL corriente que se encuentra el usuario. Una consulta tiene la siguiente estructura:

searchModeSelect
from : tipo de objeto
where : predicado

searchMode: define el espacio de búsqueda y puede ser **global,local** o **deep.**
global: el espacio de búsqueda son todos los objetos de un usuario en particular. Todos los links y nodos que el usuario tiene acceso en el momento de hacer la consulta.

deep : la búsqueda comienza en el nodo SBL corriente y se propaga recursivamente de acuerdo a la jerarquía del nodo SBL.

local : la búsqueda local sólo considera el nivel interno del nodo SBL corriente. No hay propagación de búsqueda en nodos SBL que estén dentro del nodo (o sea nodos SBL anidados).

El **tipo de objeto** puede ser AtomicNode, Link y SBLNode.

EL resultado de una consulta es un conjunto de objetos que cumple con la cláusula **where.**

Veamos 2 ejemplos sencillos:

1) Todos los nodos atómicos con nombre "hipermedia" o categoría "abstracta". Ejecutar una búsqueda local :

```
localSelect
from: AtomicNode
where : (o.name = "hipermedia" ) | (o.category="abstracta ")
```

2) Todos los g-links con categoría "principiante". Ejecutar una búsqueda en profundidad:

```
deepSelect
from: Link
where: (l isKindof : Glink ) & (l.category = "principiante")
```

2.2.3 CONCLUSION

Hemos presentado las características principales de Hydesign. Este modelo extiende el modelo de datos básico de nodos y links. Se han introducido diferentes tipos de nodos SBL como también distintos tipos de links de agregación, los cuales dan una mayor flexibilidad al diseñador para armar estructuras de datos. Los nodos SBL básicamente representan ambientes locales dentro de la red del hipermedia y dan soporte para la modularización de un diseño. Anidando nodos SBL podemos construir distintas capas o niveles de abstracción.
Con los distintos tipos de links de agregación se modelaron distintos tipos de grafos (grafos dirigidos, jerarquías y secuencias) que se usarán según las necesidades de cada aplicación.

Para terminar enunciaremos algunas ventajas y desventajas de Hydesign:

Ventajas:
1) Es un modelo orientado a objetos; esto significa que para las personas que ya tienen idea de lo que es este paradigma, se les facilita la comprensión del modelo Hydesign. Y obviamente que contamos con todas las ventajas de un modelo de diseño orientado a objetos[*REBE*90].

2) Tanto los nodos como los links son tipados. Esto permite agregar atributos a los mismos , los cuales definen a un tipo como tal.

3) Ofrece la posibilidad de introducir el concepto de composición para extender el modelo base de nodos y links (que sabemos que ya no es suficiente para modelar ciertas aplicaciones hipermediales). Hydesign hace una clara distinción entre los distintos tipos de estructuras o composiciones.

4) Introduce un lenguaje de consultas, aunque muy primitivo.

Desventajas:
1) Para representar el contenido o información que tiene un nodo, disponemos del atributo *content*. Desde el punto de vista del modelo, no se permite especificar en forma más detallada o descomponer la estructura del objeto que tendrá este atributo. Para ciertas aplicaciones, este aspecto del modelo puede representar una desventaja.

2) Si bien se presenta un lenguaje de consultas SQL-like, este no es lo suficientemente expresivo, ya que por ejemplo, no se pueden especificar caminos o sub-redes a través de expresiones regulares sobre los tipos de los nodos o links. Debido a esta limitación, tampoco contamos con la posibilidad de tener visiones o estructuras virtuales determinadas por la respuesta a una consulta. Como dijimos anteriormente, Hydesign es principalmente un modelo de diseño, sobre el cual se pueden hacer consultas muy restringidas.

3) Un mismo nodo no puede pertenecer a 2 agregados de hipermedia diferentes. Ejemplo : un nodo no puede pertenecer a una secuencia y a una jerarquía al mismo tiempo.

2.3 02SQL : UN LENGUAJE DE CONSULTAS

En esta sección se presentan 2 herramientas que sirven a un determinado fin : especificar una aplicación y luego poder consultarla[*RIZK*94]. Una de ellas es un lenguaje que sirve para especificar aplicaciones, su nombre es O2. La otra es un lenguaje de consultas que sirve para consultar aplicaciones especificadas en O2. Empecemos con una breve descripción de este lenguaje.

2.3.1 02 : UN LENGUAJE PARAR ESPECIFICAR UNA APLICACION

Presentaremos un ejemplo de un conjunto de clases que sirven para definir un artículo. Analizando este ejemplo tendremos una idea general de este lenguaje. Esta especificación está hecha usando el lenguaje 02[*O*293] y las consultas se harán usando este ejemplo.

```
class Artículo type tuple      (título:Title,
                                autores:list(Autor),
                                abstract:Abstract,
                                secciones:list(Sección),
                                resumen:Resumen)
                  constraint   : título!=nil,autores!=list(),abstract!=nil,sections!=list()
class Título                   Text
class Autor                    Text
class Abstract                 Text
class Resumen                  Text
class Sección   type  union    (a1:tuple(título: Título,contenidos:list(Contenido)),
                                a2:tuple(título:Título,contenidos:list(Contenido),
                                      subsecciones:list(Subsección)))
                  constraint   : (a1.título!=nil,a1.contenidos!=list()) |
                                 (a2.título!=nil,a2.subsecciones!=list())
class Subsección type tuple    (título:Título,contenidos:list(Contenido),label:string)
                  constraint   : título!=mil,contenidos!=list()
class Contenido   type union   (figura:Figura,párrafo:Párrafo)
                  constraint   : figura!=nil | párrafo!=nil
class Figura   type tuple      (label:string,sizex:string,sizey:string)
                  constraint   :  (sizex!=nil)
class  Párrafo type tuple      (reflabel :union(subsecc:Subsección,
                                                figura   :Figura))
                  constraint   : (reflabel.subsecc!=nil | reflabel.figura!=nil)
name Artículos list(Artículo)
```

Figura 7 : clases para entidades del tipo Artículo

Aclaración: en la figura 7 no se escribió una especificación completa como para ser compilada, sólo escribimos lo necesario como para analizar este fragmento desde el punto de vista de las consultas que se pueden hacer (ya que éste es el tema de nuestro interés) . Con esto logramos esconder los detalles que son irrelevantes y sólo contribuyen a distraer la atención del lector.

En la figura 7 podemos observar 2 constructores usados para definir las clases : **tuple** y **union. Tuple** sirve para expresar una lista de atributos que definen una clase. El otro constructor es **union** y tiene el mismo significado que se le da en los lenguajes de programación. Sirve para indicar que el tipo puede ser definido como alguna de las posibles formas especificadas (en el ejemplo la primer forma se nombra con a1 y así sucesivamente).

También tenemos el constructor **list** que define una lista de elementos. El tipo de los elementos de la lista se indica entre los paréntesis que hay siguiendo a la palabra list.

La palabra clave **constraint** es para agregar semántica a una definición de clase e indica condiciones sobre los atributos de una clase (por ejemplo : a1.contenidos!=list() significa que el atributo contenidos de la clase sección no puede ser una lista vacía).

2.3.2 O2SQL : UN LENGUAJE DE CONSULTAS PARA 02

Este lenguaje de consultas [*CLUE*89] fue diseñado para ser usado sobre una especificación hecha en 02. Este lenguaje tiene su "fuerte" cuando se lo usa para consultar bases de datos de texto donde la información no tiene estructura plana y se necesita hacer consultas sobre la estructura lógica que posee un objeto o entidad.

Un constructor que sale de lo común es PATH. El valor de este constructor es un camino concreto a través de objetos complejos. Por ejemplo mi_artículo.secciones[1].subsecciones[0].título es un camino concreto. Este camino selecciona el atributo sección de mi_artículo (la segunda sección), la primer subsección de esta sección y el título de esta subsección. Los índices corresponden al los respectivos elementos de la listas.
En una consulta las variables camino se distinguirán por tener la palabra clave PATH_ precediéndolas.
A continuación veremos distintos tipos de consultas hechas en este lenguaje:

C1 : *Buscar el título y abstract de los artículos que tienen una sección con título conteniendo la palabra "hipertexto"*

```
select : tuple (título : a.título , abstract:a.abstract)
from   : a in Artículos , s in a.secciones
where : s.títlulo contains "hipertexto"
```

El predicado contains permite evaluar si una cadena de caracteres está contenida en otra.

C2 : *Buscar todos los títulos en el artículo : mi_artículo*

```
select : t
from   : mi_artículo.PATH_p.título(t)
```

Acá estamos buscando en un artículo en particular llamado mi_artículo.
El resultado es un conjunto de títulos alcanzados desde la raíz de mi_artículo siguiendo los caminos posibles hasta encontrar un título.
En esta consulta la expresión mi_artículo.PATH_p.título(t) ilustra un nuevo tipo de consultas con respecto a lo tradicional. Es un camino con 2 variables (PATH_p y t), cuya semántica difiere de la consulta anterior (C1). En C2 se retorna un conjunto de tuplas con 2 atributos : t y PATH_p. EL atributo PATH_p varía sobre todos los caminos concretos empezando desde la raíz (mi_artículo) y terminando sobre el atributo título. El valor del atributo t en la tupla (t,PATH_p) corresponde al título que puede ser alcanzado desde mi_artículo siguiendo el camino concreto denotado por el valor del atributo PATH_p.

Acompañando el constructor PATH tenemos algunas funciones como **length**. Por ejemplo supongamos que P es un camino concreto cuyo valor es secciones[0].contenidos[0], la longitud de P es 2 y podemos proyectar P sobre sus elementos : P[0] = secciones[0] . Para ilustrar esta función veamos la siguiente consulta :

C3 : *Encontrar todos los hijos y nietos de mi_artículo*

```
select : x
from   : mi_artículo.PATH_p(x)
where  : length(PATH_p)<=2
```

El resultado de la expresión mi_artículo.PATH_p(x) y la condición, es un conjunto de caminos alcanzados desde la raíz de mi_artículo hasta 2 pasos hacia abajo en la jerarquía. La variable x en la cláusula select retorna los elementos correspondientes (es decir, los hijos y nietos a partir de mi_artículo).

C4: *Encontrar todas las hojas de mi_artículo*

```
select : x
from   : mi_artículo.PATH_p(x), x.PATH_q
where  : length(PATH_q)=0
```

En C4 la expresión x.PATH_q retorna el conjunto de caminos concretos alcanzados desde un elemento x en la estructura de mi_artículo.

2.3.3 NAVEGANDO POR LOS LINKS

Si vemos la figura 7 y la definición de la clase párrafo, podemos notar que el atributo reflabel es una referencia a una figura o subsección. Veamos cómo usamos estas referencias en las consultas:

C5 : *Encontrar todas las figuras referenciadas dentro de una sección de un artículo por un párrafo que contiene la frase "objeto complejo"*

```
select : c.reflebel.figura
from   : a in Artículos, s in a.secciones, c in s.contenidos
where  : c contains "objeto complejo"
```

Recordemos que según el esquema denotado por la figura 7, las secciones son una unión : una sección marcada con a1 corresponde a una tupla con atributos título y contenidos, y una sección marcada con a2 corresponde a una tupla con atributos título, contenidos y subsecciones. Los contenidos también son una unión, ya que pueden ser una figura o un párrafo.

Para omitir marcadores de unión en la cláusula from (por ejemplo s.a1.contenidos o s.a2.contenidos) y evitar un fallido al evaluar la cláusula where (por ejemplo contenidos que no tienen un párrafo) , se ha introducido el concepto de "selectores implícitos". Cualquier operación sobre una variable que varía sobre el dominio de un tipo unión implica una selección implícita. También en O2SQL se puede usar la expresión c contains "objeto complejo" en la cláusula where para restringir c a tipo texto (en el ejemplo : párrafos). Finalmente desde que en nuestro esquema definimos que los párrafos pueden tener

referencias a figuras o secciones, la expresión c.reflabel.figura selecciona explícitamente a las figuras.
Debemos aclarar que la expresión **c.reflabel** en la cláusula select hace posible la navegación a través de las referencias cruzadas entre los elementos de la estructura del documento.

C6 : *Encontrar los elementos que tienen una referencia a la segunda subsección de la primer sección de mi_artículo.*

```
select: x
from  : mi_artículo.PATH_p(x)
where: x.reflabel = mi_artículo.secciones[0] .subsecciones[1]
```

La expresión mi_artículo.PATH_p(x) en la claúsula from selecciona todos los elementos de mi_artículo. Pero x.reflabel en la cláusula where es usado para restringir la variable x a los elementos de mi_artículo que tienen el atributo reflabel definido (en nuestro caso : párrafo) y apuntan a una subsección.
Finalmente la expresión mi_artículo.secciones[0].subsecciones[1] es usada para seleccionar la segunda subsección de la primer sección de mi_artículo.

La siguiente consulta ilustra la navegación sobre la estructura del documento sin tener conocimiento preciso de los links que hay en el mismo:

C7 : *Encontrar los elementos que tienen una referencia a un elemento dentro de la primer sección de mi_artículo*

```
select : x
from  : mi_artículo.PATH_p(x) , mi_artículo.secciones[0] .PATH_p(y)
where : x.reflabel = y
```

En este caso se devuelven los párrafos que tienen una referencia a una subsección o figura en la primer sección de mi_artículo.

2.3.4 CONCLUSIONES

Lo que hemos presentado anteriormente es un nuevo tipo de consultas SQL-like .O2, es un lenguaje para especificar un esquema, pero no podemos decir que es un modelo para construir aplicaciones hipermediales, como lo sería HYDESIGN. O2 es simplemente un lenguaje donde se pueden definir clases de objetos con la flexibilidad de poder especificar las entidades con un gran detalle de descomposición (que esta es la falencia de HYDESIGN, ya que el atributo que representa el contenido de un nodo no puede ser especificado con gran detalle en el modelo, para más explicaciones ver la sección 2.2.1).
El tipo de consultas que se pueden realizar con O2SQL tiene su "fuerte" cuando se las usa sobre aplicaciones **muy específicas** donde interesa definir en forma muy detallada la estructura lógica interna de una entidad u objeto y sobre todo : **deseamos hacer consultas sobre esta estructura lógica interna.**

El constructor **PATH** provee un medio flexible de hacer consultas, pero si proyectamos sobre una variable que esté involucrada en este constructor, no sabemos de antemano el tipo de los elementos que van a formar la respuesta. También puede ser que la respuesta esté formada por elementos de distintos tipos (algo que puede confundir al usuario), ya que PATH es un constructor que se puede instanciar con caminos concretos que pueden tener longitudes distintas (esto se aprecia en las consultas C3 o C4).

Una **desventaja** de esta especificación en O2 es que no se pueden definir los links como verdaderas entidades con atributos. Es decir no se contempla el caso de tener una clase Link predefinida (standard en O2) y estudiar como ésta serviría para crear links navegacionales entre objetos.
Una **ventaja** para destacar es que en este nuevo tipo de consultas SQL-like, se pueden incluir tanto atributos de las entidades en la cláusula from (esto sugiere que un atributo puede ser visto como una entidad) como objetos u entidades (como mi_artículo en varias de las consultas). Esto último podría abrir un nuevo camino en los tipos de consultas SQL-like.

CAPITULO 3

UN MODELO PARA CONSULTAR APLICACIONES HIPERMEDIALES

CONTENIDO

En este capítulo desarrollamos el modelo en que nos basamos para construir más tarde un servidor de consultas. En la primer sección veremos como nace este modelo. La segunda sección la dedicamos a una componente del modelo : los links, que son tratados como verdaderas entidades. En la tercer sección describimos las consultas que se pueden hacer en este modelo. En la cuarta sección damos definiciones formales de los elementos del modelo. En la quinta sección describimos el Álgebra de Hiperwalks, que es el álgebra en que se basan las operaciones intervinientes en una consulta. Por último, en la sexta sección hacemos una conclusión acerca de este modelo, que nos da la base para en el capítulo siguiente hacer una extensión al mismo.

3.1 COMO NACE EL MODELO

Para entender esto, debemos mencionar que hay por lo menos 2 aplicaciones que conviven para realizar una consulta. Una de ellas, un servidor de consultas (SC, para abreviar) y la otra una aplicación hipermedial (AH,en adelante). Este uso que hace la AH del SC es con el objetivo de que la última resuelva todo lo que concierne a las consultas originadas por el usuario que se encuentra navegando en la AH. Gráficamente tenemos:

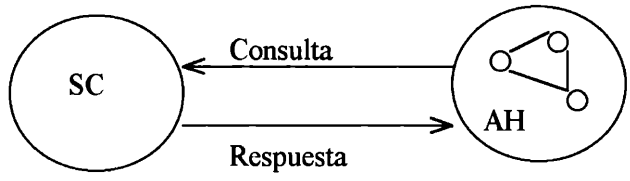


Figura 8

En una primer parte veremos que condiciones deben cumplir las aplicaciones hipermediales que deseen utilizar al SC. Para ello en esta sección y la siguiente explicaremos el modelo en que nos basamos : el modelo GRAM [*AMMA*92], que es un modelo que sirve para especificar y principalmente consultar AH.

Para empezar diremos que la aplicación que desee interactuar con el servidor de consultas debe ser una aplicación de hipermedia. En una primera aproximación esto significaría que su dominio de datos pueda ser representado por el modelo de nodos y links. Un ejemplo típico de estas aplicaciones son los hiperdocumentos, donde se tienen nodos para representar capítulos, secciones, subsecciones, etc. y links para conectar estas componentes de un documento. En este tipo de aplicaciones los modelos comunes de datos son a menudo ineficientes y no capturan la estructura de los datos que representan a un hiperdocumento [*MEND*89] y [*MEND*90] .

En general, para cualquier aplicación hipermedial, podemos decir que los nodos representan objetos o entidades del mundo real y los links establecerían las relaciones que existen entre los objetos.

Ahora bien , cuando estamos en una aplicación en particular esta posee un determinado dominio, que serían los objetos que forman parte de la misma. Estos objetos pueden ser clasificados según sus características en común, formando tipos (o clases) de objetos (análogamente a lo que significa una clase en programación orientada a objetos). También podemos observar que las relaciones que existen entre los objetos pueden ser clasificadas (por ejemplo: entre 2 secciones consecutivas de un capítulo existe la relación 'siguiente' , mientras que entre una determinada sección y un capítulo existe la relación 'pertenece'). Podemos ver el dominio de la AH representado con un grafo, donde los nodos representan objetos (cada nodo tendrá un tipo asociado) y los links representan las relaciones que existen entre ellos (estos también con un tipo asociado) [*HALA*90]

A continuación presentamos un ejemplo para dejar en claro estos conceptos (nodos, links y tipos). Este ejemplo también nos servirá para una posterior discusión acerca de los tipos

intervinientes en una aplicación. Se trata de una agencia de turismo que organiza viajes , los cuales tienen paradas en distintas ciudades :

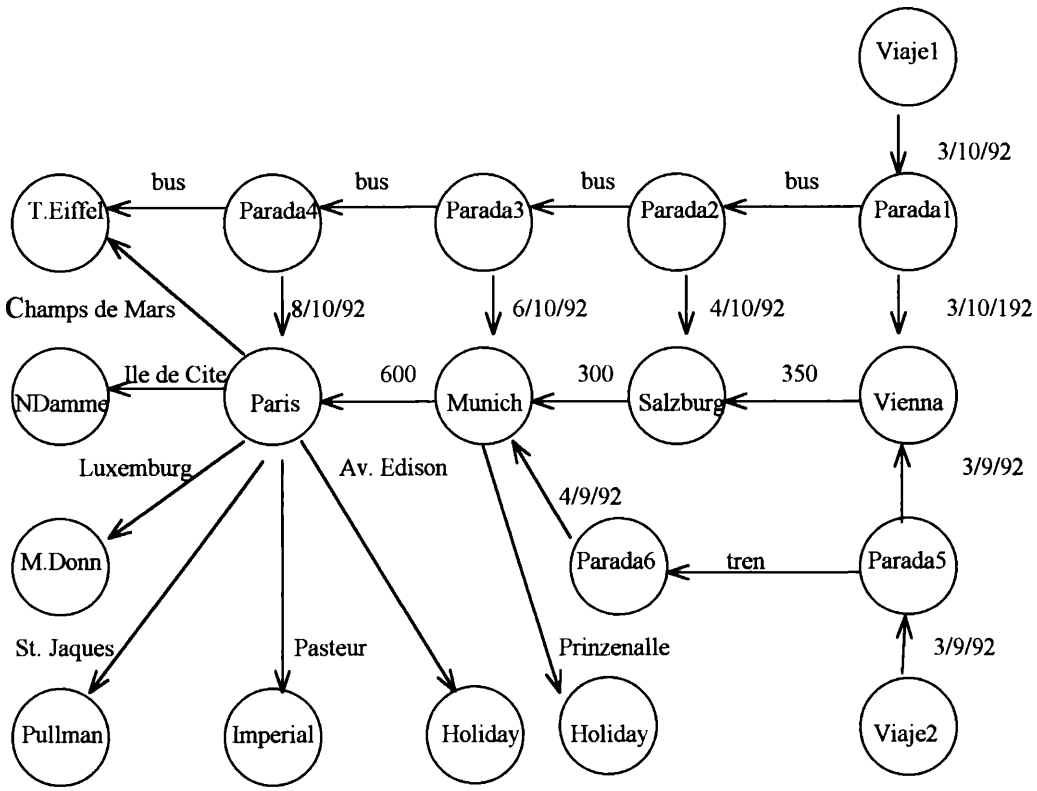


Figura 9

Como podemos apreciar tenemos labels asociados a los nodos. Estos labels corresponden a nombres de ciudades (Vienna , Paris , etc.) , nombres de paradas (Parada1, Parada2, etc.), y demás nombres. Estos nodos tienen asociado un tipo (por ejemplo: Vienna y Paris pertenecen al tipo CIUDAD, mientras que Parada1 y Parada2 pertenecen al tipo PARADA). También los links tienen labels, como nombres de direcciones (Pasteur, Av. Edison, etc.) , fechas , o distancias. Cada uno de estos links también tienen un tipo asociado. Por lo tanto podemos mencionar la existencia de un conjunto de tipos asociado al grafo, T = { CIUDAD, PARADA, VIAJE, HOTEL, RESTAURANT, MONUMENTO, dirección, distancia, primera, cuando, en }. Teniendo la información de los tipos podemos formar un grafo que representará el esquema de la AH:

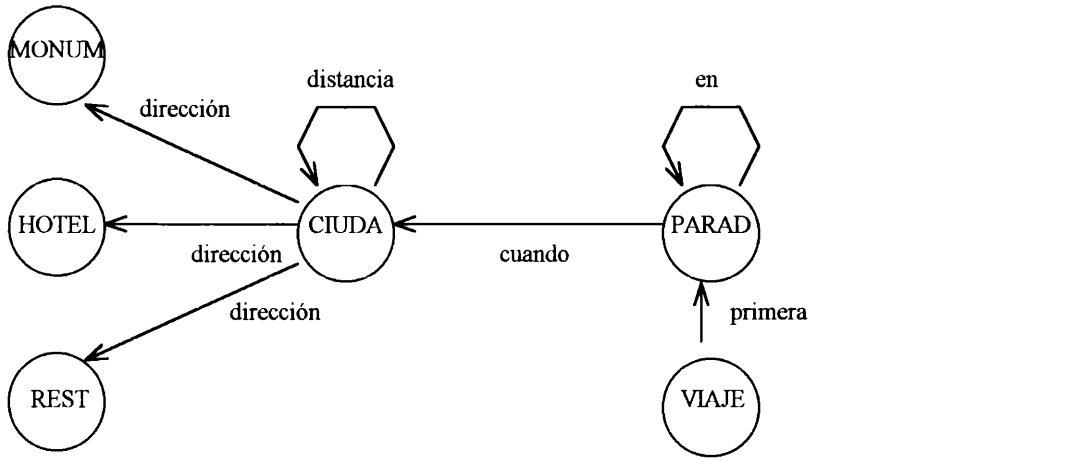


Figura 10

En conclusión podemos decir que la AH va a tener un ESQUEMA, donde se refleja la estructura de grafo que tendrá su dominio (junto con los tipos de nodos y links existentes en la misma). Intuitivamente sabemos que para un esquema , hay varias instancias, una de ellas es la presentada en la figura 9.

3.2 TIPOS DE LINKS.

Con respecto a los tipos de links, podemos decir que lo que caracteriza principalmente a un determinado tipo de link es los tipos de nodo que une, es decir el tipo del nodo origen y el tipo del nodo destino. Si tenemos el conjunto de links que son de un determinado tipo de link, lo que uno espera normalmente es que todos esos links tengan el mismo tipo de nodo origen y el mismo tipo de nodo destino. A continuación enunciaremos una propiedad asociada a lo dicho anteriormente:

Propiedad de uniformidad de tipos: Sea TL = {TL1 , TL2 , ... , TLN} el conjunto de tipos de links que existen en el esquema de una aplicación. Diremos que este esquema cumple con la propiedad de uniformidad de tipos $\Leftrightarrow$

TIPO(ORIGEN(L1))=TIPO(ORIGEN(L2))=....=TIPO(ORIGEN(LN)) y
TIPO(DESTINO(L1))=TIPO(DESTINO(L2))=.....=TIPO(DESTINO(LN))

siendo TLi= { L1,L2, ..., LN} el conjunto de links del tipo de link TLi , $\forall$ i = [1..n]

ORIGEN(Li): devuelve el nodo origen del link Li
DESTINO(Li): devuelve el nodo destino del link Li
TIPO(Ni) : devuelve el tipo del nodo Ni

Si vemos la figura 10 vemos que este esquema no cumple con al propiedad de uniformidad de tipos , ya que el tipo del link dirección une tanto nodos de tipo CIUDAD con nodos de tipo HOTEL, RESTAURANT y MONUMENTO. Por lo tanto no se cumple la segunda parte de esta propiedad , ya que (TIPO(DESTINO(Pasteur))=HOTEL) $\neq$ (TIPO(DESTINO(Luxemburg))=RESTAURANT).

Con respecto a los tipos (tanto de nodos como de links) , tienen asociado un conjunto de atributos que los caracteriza (por ejemplo el tipo CIUDAD tendrá los atributos cant_habit , fundador , etc., cada uno de ellos con un determinado dominio).
En esta parte hemos determinado conceptualmente cuales son las condiciones que debe cumplir la AH para poder interactuar con el SC, en la siguiente parte veremos que tipos de consultas podrá responder el SC.

3.3 CONSULTAS

Según [*HALASZ87*] podemos clasificar las consultas en dos grupos:

1- **Content-based querys** (consultas basadas en el contenido o semántica de la AH) : que sería buscar aquellos nodos en la AH que contienen cierto pattern de información (este pattern podrá ser una cadena de caracteres si nos encontramos en un hipertexto o podrá ser un sonido , video u otro tipo de información si estamos en una aplicación de hipermedia).

2- **Structure-based querys** (consultas basadas en la estructura del grafo de la AH) : que independientemente del contenido o semántica del AH, nos basamos en encontrar caminos en el grafo de la AH que satisfagan un pattern (o estructura) determinado.

El SC a desarrollar hará énfasis en este último tipo de consultas, es por ello que dedicamos la parte anterior a definir como debe ser, o que modelo debe seguir, la estructura del grafo de la AH.

Comenzaremos dando ejemplos de consultas que se podrán efectuar, desde las más simples hasta las más complejas.
Supongamos que queremos saber en que ciudades hay un hotel Holiday y un restaurant M. Donnalds. Observando el grafo del a base de datos (figura 9) veremos que el subgrafo :

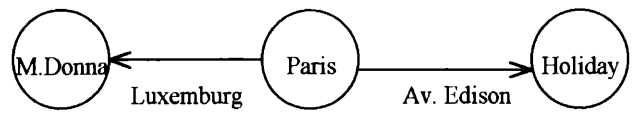


Figura 11

estaría involucrado en la respuesta a esta consulta. Este subgrafo lo vamos a denominar hiperwalk, que sería el equivalente a una tupla del Álgebra Relacional, el esquema al que obedece este hiperwalk es: CIUDAD dirección RESTAURANT + CIUDAD dirección HOTEL.

Siguiendo una sintaxis a la SQL , tendríamos la siguiente consulta:

```
select CIUDAD
from   CIUDAD dirección RESTAURANT + CIUDAD dirección HOTEL
where RESTAURANT.nombre = 'M. Donnalds' and HOTEL.nombre = 'Holiday'
```

A continuación explicaremos brevemente cada una de las tres componentes de la consulta:
_from: determina el conjunto inicial de hiperwalks sobre el cual se trabajará. Si r = CIUDAD dirección RESTAURANT + CIUDAD dirección HOTEL , entonces llamaremos I(r) a este conjunto.
_where: esta componente especifica el criterio de selección que tendremos sobre el conjunto I(r); es decir con que hiperwalks de I(r) nos quedaremos. Esta condición será establecida sobre los atributos de los tipos involucrados en la cláusula from.
_select: acá especificaremos con que componentes de los hiperwalks seleccionados nos quedaremos, es decir sobre que componente vamos a proyectar los hiperwalks. Si bien esta operación se llama 'proyección', no es tan simple como lo sería en el Álgebra Relacional, donde simplemente especificamos los atributos del esquema sobre el cual se proyectará. En este nuevo álgebra (Álgebra de Hiperwalks), para poder proyectar se deben cumplir ciertas condiciones que veremos en la sección siguiente.

Nota: Cuando demos definiciones formales en la sección siguiente seguiremos hablando de estas componentes de una consulta, por ahora sólo consideramos suficiente una introducción.

La siguiente consulta da toda la información de recorridos que incluyen una visita al monumento St. Michel en Marzo:

```
Select *
From   VIAJE primera PARADA en ) PARADA cuando  CIUDAD dirección
          MONUMENTO
Where cuando.mes='Marzo' and MONUMENTO.nombre = 'St. Mitchell'
```

El * significa que no se proyecta, que nos quedamos con toda la información de los hiperwalks. La cláusula Where especifica una condición sobre el tipo de link cuando.
Lo más provechoso para destacar en esta consulta es el contenido de la cláusula From, la cual involucra un tipo de link seguido de un paréntesis : en). Esto da origen al concepto de recursión en las consultas , y probablemente este concepto hace que este lenguaje de consultas obtenga un poder de expresividad considerable. Al escribir VIAJE primera PARADA en) PARADA cuando CIUDAD dirección MONUMENTO estamos diciendo que el conjunto I(r) serán aquellos hiperwalks que sean de la forma :

```
VIAJE primera PARADA cuando CIUDAD  dirección MONUMENTO  U
VIAJE primera PARADA en PARADA cuando  CIUDAD dirección MONUMENTO  U
VIAJE primera PARADA en PARADA en PARADA cuando CIUDAD dirección
  MONUMENTO
U .....
```

En general si tenemos una expresión con recursión NL)N, donde N es un tipo de nodo y L es un tipo de link recursivo, tendremos que:

$I(NL)N) = U\ I(NLiN)$ siendo $NL0N = N$
$i=1..\infty$ $NL1N = NLN$
$NL2N = NLNLN$ y así sucesivamente

Observación : notar que ponemos el signo) detrás del tipo de link para indicar que hay recursión, pero lo que se repite es el link y el nodo anterior.

3.4 DEFINICIONES FORMALES DE LOS ELEMENTOS DEL MODELO

Definición : un esquema de una base de datos expresada como grafo es un multigrafo , dirigido , con labels y weakly-connected*:
S=(Ns,Es,Js,Ms,T) donde :
1- Ns es un conjunto de nodos
2- Es es un conjunto de links
3- Js es la función de incidencia $Js : Es \rightarrow Ns \times Ns$
4- T = Tn U Te es el conjunto de labels (serían los tipos de nodos y links)
5- Ms es la función de asignación de labels a nodos y links , $Ms : Ns\ U\ Es \rightarrow T$

Adicionalmente S debe cumplir las siguientes propiedades :
a- Nodos distintos deben tener labels distintos
Si $n1 <> n2 \Rightarrow Ms(n1) <> Ms(n2)$

b- Al ser S un multigrafo , un par de nodos puede estar conectado por más de 1 link. Si 2 links conectan el mismo par de nodos en la misma dirección, entonces estos links deben tener distintos labels :
$Js(e) = Js(e')$ y $e <> e' \Rightarrow Ms(e) <> Ms(e')$

Nota* : weakly-connected significa que el equivalente grafo no dirigido está conectado.

Ahora bien , cada símbolo t en T es un tipo , que tiene asociado un conjunto de atributos. Dentro de este conjunto de atributos debemos nombrar un atributo distinguido o clave. Este atributo es el que mejor identifica al tipo como tal (por ejemplo, el tipo CIUDAD puede tener nombre y superficie entre otros atributos, pero nombre sería una buena elección para atributo distinguido, ya que principalmente una ciudad se identifica por su nombre más allá de su superficie u otros atributos como número de habitantes). Definimos V , como la unión de los dominios de los atributos distinguidos de todos los tipos en T ; V = U dom(Atr_dist(t)) para todo t de T. Atr_dist() devuelve el atributo distinguido de un tipo.

Resumiendo diremos que la AH deberá tener un esquema de grafo que respete el modelo de grafo presentado anteriormente : S. Ahora definiremos el grafo de la base de datos en sí. Igualmente que en Álgebra Relacional, podemos hablar de una base de datos que respete o pertenezca a un cierto esquema :

Definición : una base de datos (gráfica) con esquema S es un multigrafo dirigido y con labels:
G = (N, E , J , M ,V) donde :

1- N es un conjunto de nodos
2- E es un conjunto de links
3- V es un conjunto de labels (V como se lo definió anteriormente, cada valor es un label)
4- J es la función de incidencia J : E → NxN
5- M es la función de asignación de labels , M : N U E → V

y existe una función v de G a S , asociando a cada nodo (link) en G con un nodo (link) de S tal que:
a- v retorna para cada nodo g de G , un nodo s de S tal que el label de g está en el dominio del label de s :
$M(g) \in dom(M(v(g)))$

b- v retorna para cada link l en G, que va de un nodo g en g', un link h de S, que va de un nodo v(g) en v(g') y el label de l está en el dominio del label de h:
$Js(v(l)) = (v(g),v(g'))$ y $M(l) \in dom\ (Ms(v(l)))$

Informalmente hablando, una instancia de un esquema de base de datos S, es una base de datos gráfica. Tenemos una función de instanciación I que asocia a cada nodo (link) s de S, un conjunto de nodos (links) en G de la siguiente forma :
$I(s) = \{ g / v(g)=s \text{ y } g \in G\}$,
en el ejemplo I(PARADA)={Parada1, Parada2, Parada3, Parada4, Parada5, Parada6}

Notar que no se exige que G sea un grafo conectado. Se pueden tener nodos sin links o más de un link saliendo de o entrando a un nodo . Aquí tenemos un ejemplo de ello :

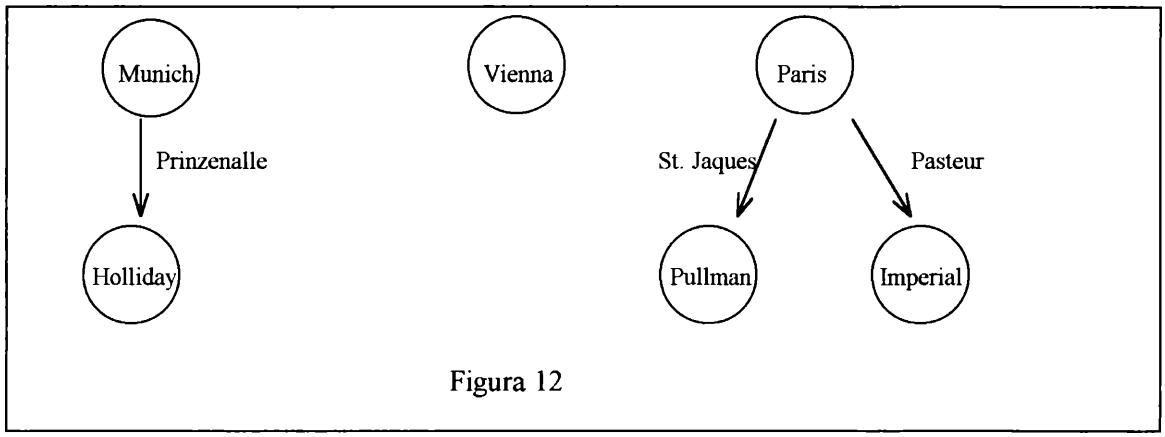


Figura 12

Como la AH tiene un esquema de base de datos, la base de datos debe respetar ese esquema. Ahora definiremos formalmente que es lo que debe ir en la cláusula From de una consulta.

3.4.1 WALKS e HIPERWALKS

WALKS (caminos) son los objetos básicos de este modelo. Un walk en un grafo es una secuencia alternada de nodos y links n0e0n1e1.....ni, comenzando y terminando en un nodo, en donde cada link es incidente con el nodo inmediato precedente y siguiente a ese link [*HARA71*]. Por ejemplo para llegar a las ciudades de un viaje , uno elige los walks en la base

de datos que empiezan con un nodo del tipo VIAJE y terminan con un nodo del tipo CIUDAD. Un ejemplo sería:

Viaje1.3/10/92.Parada1.Bus.Parada2.4/10/92.Salzburg

Generalmente, los walks que uno hace en cierto momento o situación, pueden estar relacionados con la información que tienen otros walks. Llamemos hiperwalk a un conjunto de walks relacionados (la forma en que deben estar relacionados estos walks la veremos más adelante). Por ejemplo podríamos tomar los walks que empiezan en distintos viajes y terminan en ciudades iguales. La figura anterior muestra un hiperwalk de 2 walks : uno conectando PARIS con el hotel PULLMAN y el otro walk conectando PARIS con el hotel IMPERIAL.

3.4.2 WALK_EXPRESSIONS e HIPERWALK_EXPRESSIONS

Un walk_expression (we) es una expresión regular (er) sobre T, sin alternación (+) , el contenido de un we sólo será un secuencia alternada de tipos de nodos y tipos de links, empezando y terminando con un tipo de nodo.

Informalmente podríamos decir que un we define varios walks (estos walks tendrán la estructura de tipos del we). Por ejemplo PARADA en) PARADA cuando CIUDAD dirección HOTEL es un we y uno de los walks que satisface su estructura es el walk :

Parada4.8/10/92.Paris.Pasteur.Imperial

Definición 3.4.2:

Un hiperwalk-expression (hwe) es una expresión regular sobre T que cumple 2 condiciones:

1) se puede reescribir como una suma de we : r = $\sum_{i=1..n} ri$ tal que

2) el siguiente grafo no dirigido con labels G(r) esta conectado:

G(r) = (N,E,Mg) lo formamos así:

a) por cada ri tendremos un nodo ni en N con el label ri (Mg(ni) = ri) y

b) existe un link con label t entre dos nodos nj y nk $\Leftrightarrow$ t es un tipo que esta en rj y rk.

Si luego de formar G(r) de acuerdo a la forma descripta nos queda un grafo conectado , entonces esa expresión regular es un hiperwalk-expression.

Esta condición (2) de conectividad es necesaria para asegurar que los walks de un hiperwalk estén relacionados unos con otros. Informalmente podemos decir que si en algún momento tenemos información (we) que queremos relacionarla porque comparten cosas (tipos de nodos), entonces esa forma de relacionar estos we es a través de un hiperwalk-expression.

Ahora veremos formalmente cuando un hiperwalk satisface un hiperwalk-expression, pero para ello primero debemos ver como se obtiene el label de un walk y un hiperwalk:

_El label de un walk se obtiene sustituyendo cada terminal(*) por su correspondiente label

M(n1e1n2e2.....e(n-1)nn) = M(n1)M(e1).....M(e(n-1))M(nn)

_El label de un hiperwalk h se obtiene reemplazando cada walk en h por su correspondiente label:

M({w1,w2,w3,...,wn)} = {M(w1),M(w2),M(w3),.....,M(wn)}

_Ahora definimos un lenguaje sobre T

L(t) = dom(t) (para todo t de T)
L(AB) = L(A)xL(B) (siendo A y B er)
L(A+B) = L(A) U L(B) (siendo A y B er)
L(A)) = U L(Ai) para i>=0 siendo L(A0) = {@} (vacío) (siendo A er) .Para entender el concepto de recursión ir a la sección 3.3

Aclaración *: un terminal es un nodo o un link , las e's son los links y las n's son los nodos.

Sea r un hwe ahora diremos que L(r) es el lenguaje de r.
Ejemplo: tomemos el walk w=Parada1.3/10/92.Vienna (los puntos entre los terminales los ponemos por cuestiones de claridad).Vienna es una instancia de CIUDAD y el walk está en L(PARADA cuando CIUDAD) : Parada1.3/10/92.Vienna ∈ L(PARADA cuando CIUDAD).

Definición 3.4.2: Un hiperwalk h = {w1,w2,w3,....,wn} satisface una hwe r (h |=r) ⇔
1) Existe una descomposición de r, tal que cada walk wi en h tiene su label en L(ri) : M(wi) ∈ L(ri).
2) Para cada tipo de nodo t compartido por ri y rj, wi y wj comparten al menos un nodo de tipo t.
Esta última condición obliga a que los walks de un hiperwalk h formen un grafo conectado: a cada link l en G(r) con label t entre dos nodos con label ri y rj corresponden 2 walks wi y wj en h tal que:
_M(wi) ∈L(ri) y M(wj) ∈ L(rj)
_wi y wj comparten al menos un nodo n que su label está en el dominio de t : M(n) ∈ dom(t)

Ejemplo: h={Viaje1.3/10/92.Parada1, Parada1.Bus.Parada2} es un hiperwalk que satisface hwe r=VIAJE primera PARADA + PARADA en PARADA. Los 2 walks en h comparten un nodo cuyo label es Parada1 y es de tipo PARADA.
{Viaje1.3/10/92.Parada1,Parada2.Bus.Parada3} no satisface r, ya que los 2 walks no comparten ningún nodo del tipo PARADA.
PARADA en) PARADA + PARADA cuando CIUDAD es satisfacido por {Parada1.Bus.Parada2.Bus.Parada3, Parada2.4/10/92.Salzburgo} , los 2 walks comparten el nodo con label Parada2.
Notar que la condición 2 establece que debe compartirse al menos 1 nodo por cada tipo t compartido entre ri y rj. Supongamos r = CIUDAD distancia CIUDAD dirección HOTEL + CIUDAD dirección HOTEL y h={Munich.600.Paris.Pasteur.Imperial, Munich.Prinzenalle.Holiday} (figura 13).El hiperwalk h no satisface r, ya que los 2 walks comparten Munich de tipo CIUDAD pero no comparten algún nodo de tipo HOTEL.

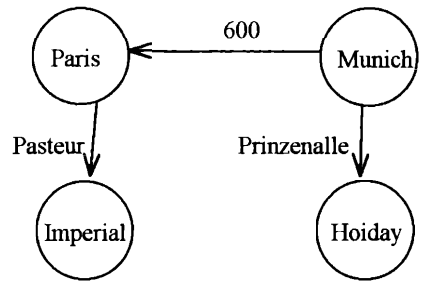


Figura 13

INSTANCIA DE UN HIPERWALK-EXPRESSION: La instancia de un hwe r en una base de datos G es definida como el conjunto de hiperwalks de G que satisfacen r:

$I(r)=\{h \mid h \in G \text{ y } h|=r\}$

Resumiendo podemos decir que la cláusula From de una consulta define este conjunto de hiperwalks. Esta cláusula recibe un hwe que determinará el conjunto de hiperwalks sobre el que luego se hará la selección y proyección.

3.5 ALGEBRA DE HIPERWALKS

El lenguaje de consultas definido se basa en el Álgebra de hiperwalks. Un consulta es una operación de la forma Y(s) o Y'(s,s') (dependiendo de si la operación es unaria o binaria) donde s y s' son conjuntos de hiperwalks y Y es una operación algebraica que es cerrada sobre el conjunto de hiperwalks. Operaciones unarias (proyección, selección, renombre) toman un conjunto de hiperwalks que satisfacen un hwe r y retornan un conjunto de hiperwalks s' satisfaciendo un hwe r' posiblemente distinto de r. Operaciones binarias (join, concatenación, operaciones típicas de conjuntos) toman 2 conjuntos de hiperwalks y retornan otro conjunto de hiperwalks.
A continuación detallaremos estas operaciones y veremos sus similitudes y diferencias con respecto al Álgebra Relacional.

1-RENOMBRE

Un hiperwalk-expression puede tener varias ocurrencias de un mismo tipo. Renombrar algunas de ellas nos permite distinguir entre las distintas ocurrencias. Por ejemplo h1={Vienna.350.Salzburgo} y h2={Munich.600.Paris} son 2 hiperwalks que satisfacen r = CIUDAD distancia CIUDAD. Para poder distinguir entre estas 2 ocurrencias de CIUDAD renombraremos la segunda ocurrencia a CIUDAD'. En CIUDAD distancia CIUDAD', CIUDAD sería la ciudad origen y CIUDAD' sería la ciudad destino. En el Álgebra relacional, el renombrado de los atributos del esquema de una relación mantiene la relación igual, es decir, no se pierden tuplas. El ejemplo dado anteriormente no se pierden hiperwalks, es decir I(r) = I(r'). Pero ésto no siempre es cierto. Analicemos el siguiente caso:
Sea r1=PARADA en PARADA , r2= PARADA cuando CIUDAD y r = r1 + r2
Si quisiéramos escribir esto como un grafo tendríamos 2 casos o alternativas:

A) donde la última componente de r1 es compartida con r2

Figura 14

o

B) donde la primer componente de r1 es compartida con r2

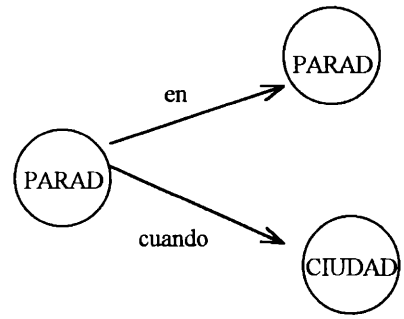


Figura 15

Por lo tanto podemos decir que los hiperwalks que satisfacen r siguen 2 formas posibles. Por ejemplo {Parada1.bus.Parada2,Parada2.4/10/92.Salzburgo} satisface la forma A o {Parada1.bus.Parada2,Parada1.3/10/92.Vienna} que satisface la forma B. Si sólo renombráramos la primer ocurrencia de PARADA en r (por ejemplo PARADA') tendríamos r'= PARADA' en PARADA + PARADA cuando CIUDAD, nos quedaríamos con un subconjunto de los hiperwalks que satisfacen r : el subconjunto que satisface A). Análogamente, si sólo renombráramos la segunda ocurrencia de PARADA en r1 nos quedaríamos con el subconjunto que satisface B).
De esto podemos decir que I(PARADA en PARADA + PARADA cuando CIUDAD) = I(PARADA' en PARADA + PARADA cuando CIUDAD) + I(PARADA en PARADA' + PARADA cuando CIUDAD).

Definición: Sean r y r' dos hwe tal que r'es obtenido a partir de r renombrando algunos tipos en r sin cambiar la definición del tipo: si t en r ha sido renombrado en t' $\Rightarrow$ dom(t) = dom(t') $\Rightarrow$ L(r)=L(r'). Sea S un conjunto de hiperwalks que satisface hwe r. Renombrando Pr'(S) mantenemos los hiperwalks de S que satisfacen r' (Pr'(S) $\subseteq$ S)

$$Pr'(S) = \{ h \mid h \in S , h|=r' \}$$

Ejemplo: h = {Parada1.3/10/92.Vienna,Vienna.350.Salzburg} satisface hwe r=PARADA cuando CIUDAD + CIUDAD distancia CIUDAD . Después de renombrar en r'=PARADA cuando CIUDAD' + CIUDAD' distancia CIUDAD, Vienna es una instancia de CIUDAD' y

Salzburgo una instancia de CIUDAD. Hiperwalk h satisface r' pero no satisface el renombramiento de r en PARADA cuando CIUDAD' + CIUDAD distancia CIUDAD'.

2-SELECCION

La selección permite evaluar funciones booleanas sobre los labels de un hiperwalk. Dentro de una expresión r podemos tener 2 tipos de subexpresiones : las simples (sin recursión) y las complejas (con recursión). En CIUDAD1 distancia) CIUDAD2 dirección RESTAURANT , CIUDAD1 distancia) es una subexpresión compleja y CIUDAD2 dirección RESTAURANT es una subexpresión simple. Si h es un hiperwalk en la instancia de r ($h \in I(r)$) entonces con una subexpresión simple v de r, tenemos asociado un único componente de h , notado h : v, que no es necesariamente un hiperwalk. Con cada subexpresión compleja S) de r corresponde un conjunto de componentes en h, cada uno de ellos satisfaciendo S. Ejemplo : h = {Salzburg.300.Munich.600.Paris} en la instancia de CIUDAD1 distancia) CIUDAD2. La componente h:CIUDAD2 es Paris y la componente h:CIUDAD1 distancia) = {Salzburg.300, Munich.600}

Definición: Las condiciones de selección sobre expresiones regulares se definen de la siguiente manera:

1-Sea t un tipo en una subexpresión simple de r. Una función booleana f(t):dom(t) $\rightarrow$ {true,false} es una condición simple sobre r.

2-Sean t y t' 2 tipos en una subexpresión simple de r. t= t' es una condición simple sobre r.

3-Si C es una condición sobre S, y S) es una subexpresión compleja de r, $\Rightarrow \exists C$ es una condición sobre r.

4-Si C y C' son condiciones sobre r, también lo son C and C', C or C', y not C.

Nota :en 2- en realidad se compararía un atributo de t con un atributo de t'.

Definición: Sea S un conjunto de hiperwalks que satisfacen r y sea C una condición sobre r. La selección sobre S con la condición C, notada OC(S), retorna los hiperwalks en S que satisfacen C:

$OC(S) = \{ h \mid h \in S \text{ y } h \text{ satisface } C\}$

Un hiperwalk satisface una condición de selección C (C es verdadera para h) si alguno de los siguientes puntos es verdadero:

1-Si C es simple , C se evalúa como si se estuvieran valuando tuplas en el Álgebra Relacional.

2-Si $C = \exists D$ y D es una condición de selección sobre s, entonces C es verdadera para h $\Leftrightarrow$ al aplicar la condición D sobre el conjunto determinado por h:s), nos queda un conjunto no vacío:

$h \models \exists D \Leftrightarrow OD(h:s))$ no es vacío.

Informalmente queremos decir lo siguiente : dado que h:s) determina un conjunto de componentes (ver el comienzo de la operación Selección), esta condición es verdadera si al aplicar la condición D (como un filtro) sobre cada uno de estos elementos, nos queda un conjunto no vacío.

3- Si $C = D$ and $F \Rightarrow h \models C \Leftrightarrow h \models D$ y $h \models F$

4- Si C = D or F $\Rightarrow$ h|=C $\Leftrightarrow$ h|=D o h|=F
5-Si C = not D $\Rightarrow$ h|=C $\Leftrightarrow$ h no satisface D.

Ejemplo: Para obtener todos los hoteles de Vienna, aplicamos OD(I(CIUDAD dirección HOTEL)), donde D(CIUDAD) = (CIUDAD.nombre = Vienna).
Sea S un conjunto de hiperwalks que satisfacen CIUDAD distancia) CIUDAD. Para obtener los hiperwalks que tienen al menos 2 ciudades vecinas que distan entre sí menos de 100 Km. tenemos:
OD(S) , donde D dom(distancia) $\rightarrow$ {true,false} es verdadera cuando distancia.metros$\leq$100.

3-PROYECCION
Informalmente hablando, la proyección de un hiperwalk h sobre un hwe r' consiste en mantener un subconjunto de walks de h o subwalks de walks de h. Por ejemplo {Parada1.bus.Parada2.4/10/92.Salzburg} es una proyección de h = {Parada1.bus.Parada2.4/10/92.Salzburg, Viaje1.3/10/92.Parada1.3/10/92.Vienna}. También es una proyección {Vienna}.

Antes de definir lo que es una proyección debemos definir 2 conceptos:
Definición: Sean r y r' 2 we. Si r es de la forma ur'v, donde posiblemente u y v sean vacíos $\Rightarrow$r' es una subexpresión de r.

Por ejemplo PARADA cuando CIUDAD es una subexpresión de VIAJE primera PARADA cuando CIUDAD. Pero PARADA en PARADA no es subexpresión de PARADA en) PARADA cuando CIUDAD.
Ahora veremos cuándo un hwe es subexpresión de otro hwe.

Definición: Sea r' un hwe. r'=$\sum_{j=1}^{m} r'j$ es una subexpresión de hwe r = $\sum_{i=1}^{n} ri$ (r'$\leq$r) $\Leftrightarrow$
1) r' es un hwe con grafo G(r') (como el definido en la definición 3.4.2).
2) m$\leq$n y para todos los walk-expressions r'j j $\in$ [1..m] existe un we ri tal que r'j $\leq$ri. ri es llamada la superexpresión de r'j.
3) Con cada link en G(r') con label t conectando n'i con label r'i a n'j con label r'j, corresponde un link en G(r) con label t conectando ni con label ri (superexpresión de r'i) a nj con label rj (superexpresión de r'j). Las ocurrencias de t en ri (rj) deben ser mantenidas en r'i (r'j).

La idea de proyección en el Álgebra Relacional es la siguiente: dada una relación (que es un conjunto de entidades = tuplas) queremos quedarnos con sólo una parte de la información que tienen estas tuplas, sólo con algunos atributos de esa relación, pero seguiremos teniendo todas las tuplas de la relación.

En el Álgebra de Hiperwalks, esto no sucede siempre así, si no proyectamos "bien" podemos perder tuplas, entonces la condición 3 surge para evitar esa pérdida de hiperwalks. A continuación veremos un ejemplo para clarificar esto:

Sea r1=VIAJE primera PARADA cuando CIUDAD1, r2=PARADA en PARADA cuando CIUDAD2 , r = r1 + r2 y h = {Parada1.bus.Parada2.4/10/92.Salzburg, Viaje1.3/10/92.Parada1.3/10/92.Vienna} que satisafce r.
Si tuviéramos r'=PARADA cuando CIUDAD1 + PARADA cuando CIUDAD2, la correspondiente componente de h es h'={Parada1.3/10/92.Vienna, Parada2.4/10/92.Salzburg} y h' no es un hiperwalk!!!. Esto es así porque no se cumple la condición 3: no se mantienen todas las ocurrencias del tipo que es compartido entre los walk-expressions. No podemos eliminar ocurrencias de PARADA porque no sabemos para cada hiperwalk que satisface r, si el PARADA que comparten sus walks es una instancia del primer o segundo PARADA de r2. En este caso r' no es subexpresión de r. Si r'=
VIAJE primero PARADA + PARADA en PARADA cuando CIUDAD, entonces $r' \leq r$ ya que mantiene todas las ocurrencias de PARADA.

Definición: Sea r' un hiperwalk-expression tal que $r' \leq r$, (r es hwe). La proyección de un conjunto de hiperwalks $S \subseteq I(r)$ sobre r', denotado IIr'(S), es el conjunto de r'-componentes, denotado h : r', del conjunto de hiperwalks h de S:

$IIr'(S) = \{h' \mid h \in S, h' = (h : r')\}$ $IIr'(S) \subseteq I(r')$

Ejemplo: La figura siguiente muestra 2 hiperwalks que satisfacen el hwe VIAJE primero PARADA en) PARADA cuando CIUDAD dirección HOTEL. La proyección sobre CIUDAD dirección HOTEL retorna {Paris.Pasteur.Imperial} y {Munich.Prinzenalle.Holiday}. Observar que la proyección sobre PARADAen)PARADA retorna {Parada1.bus.Parada2.bus.Parada3.bus.Parada4} y {Parada5.tren.Parada6}.

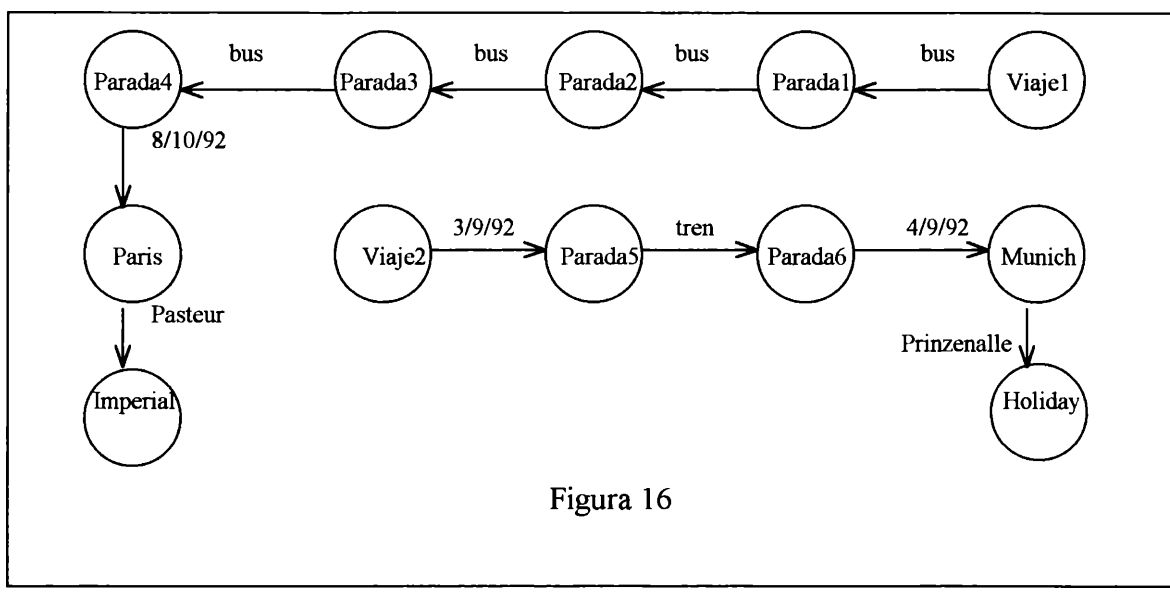


Figura 16

4-JOIN
Esta operación toma pares de hiperwalks de 2 conjuntos distintos de hiperwalks $S \subseteq I(r)$ y $S' \subseteq I(r')$ y retorna la unión que satisface hwe r + r'.

Definición: Sean S y S' 2 subconjuntos de I(r) y I(r') respectivamente. Si r + r' es un hwe $\Rightarrow$ la unión de S y S' denotada SxS', retorna un subconjunto de I(r+r'), donde cada hiperwalk del mismo es la unión de un hiperwalk de S con uno de S':

$SxS' = \{ h \cup h' \mid h \in S , h' \in S' \text{ y } h \cup h' \mid = r + r' \}$

Observar que la unión de 2 hiperwalks $h \in S$ y $h' \in S'$ no es necesariamente un hiperwalk satisfaciendo r + r'. Por ejemplo la unión de {Paris.Luxemburgo.M_Donnalds} y {Munich.Prinzenhyalle.Holiday} no es un hiperwalk. En efecto, h U h' es un hiperwalk que pertenece a SxS' si para cada par de walk-expressions ri en r y r'j en r' que comparten un nodo de tipo t, los correspondientes walks en h y h' comparten al menos un nodo de tipo t.
Ejemplo : Supongamos que tenemos un conjunto de tours $S \subseteq I$(VIAJE primera PARADA en)PARADA).Para tener sólo aquellos tours que hacen una parada en Munich, debemos hacer un join entre S y OM(I(PARADA cuando CIUDAD)) siendo M(CIUDAD) = (CIUDAD.nombre = Munich).
Si quisiéramos tener todos los tours que paran en Munich y luego en Paris, lo haríamos con la siguiente consulta : OP(I(VIAJE primero PARADA en) PARADA cuando CIUDAD))
x OM(I(PARADA cuando CIUDAD2)) siendo P(CIUDAD) = (CIUDAD.nombre = Paris) y M(CIUDAD2) = (CIUDAD2.nombre = Munich)

Podríamos decir que la operación del Álgebra Relacional que más se le parece a esta es el Join Natural.

5- CONCATENACION
Hasta ahora las demás operaciones tenían alguna similitud con las operaciones del Álgebra Relacional, pero esta es una operación propia del Álgebra de Hiperwalks, y se debe a que la misma nace de la naturaleza específica que adopta el concepto de tupla o hiperwalk = camino.

La concatenación de dos conjuntos de hiperwalks S y S' concatena un hiperwalk de S con otro de S' cuando sea posible. La idea es concatenar caminos de $h \in S$, con caminos de $h' \in S'$. Si el último nodo de un walk w de h es igual al primer nodo de un walk $w' \in h'$, entonces la concatenación de w con w', wow', se obtiene reemplazando el último nodo de w por w'. Ejemplo: Viaje1.3/10/92.Parada1 o Parada1.bus.Parada2 = Viaje1.3/10/92.Parada1.bus.Parada2. Si el último nodo de w no es igual al primer nodo de w' $\Rightarrow$ wow' = e (vacío).

La **concatenación de 2 we** se define de la siguiente manera : sean r y r' 2 we. Si r es de la forma ut y r' es de la forma tv, con $t \in Tn$ y u,v son expresiones sobre T, r0r' = utv. De otra forma utv = e (vacío).

La **concatenación de 2 hwe** $r = \sum_{i=1}^{n} ri$ y $r' = \sum_{j=1}^{m} r'j$ denotada r0r' se define como la suma de concatenaciones de walk-expressions de r con walk-expressions de r':

$r0r' = \sum (ri0rj')$ para $1 \leq i \leq n$, $1 \leq j \leq m$

Por ejemplo: PARADA cuando CIUDAD 0 (CIUDAD dirección HOTEL + CIUDAD dirección RESTAURANT) = PARADA cuando CIUDAD dirección HOTEL + PARADA cuando CIUDAD dirección RESTAURANT.

Definición : Sean S y S' 2 subconjuntos de I(r) y I(r') respectivamente.

Si r0r' <> e $\Rightarrow$ la concatenación de S con S', denotada S0S', retorna un conjunto de hiperwalks en I(r0r'), donde cada elemento del mismo es la concatenación de un hiperwalk de S con un hiperwalk de S', denotada h o h' y definida de la siguiente manera:

S0S' = { hoh' | h$\in$S , h$\in$S' y hoh' |= r0r'}

Sean h y h' dos hiperwalks satisfaciendo r = $\sum_{i=1}^{n} ri$ y r'=$\sum_{j=1}^{m} r'j$ respectivamente. La concatenación de h con h' contiene para cada par ri,r'j donde ri0r'j <> e el walk wiow'j, donde wi(w'j) de h(h') satisface ri(r'j). Observar que si ri 0 r'j <> e y wi o wj = e $\Rightarrow$ hoh' no satisface r0r'.

hoh' = { wi o wj' | wi $\in$ h , w'j $\in$ h' y wi |= ri , w'j |= r'j y ri 0 r'j <> e}

Ejemplo: dado dos conjuntos de hiperwalks S $\subseteq$ I(PARADA cuando CIUDAD) y S' $\subseteq$ I(CIUDAD dirección HOTEL + CIUDAD dirección RESTAURANT). S contiene las paradas con sus respectivas ciudades y S' tiene las ciudades con sus respectivos hoteles y restaurants. Podríamos concatenar esta información para tener los hoteles y restaurants que pueden ser visitados durante una parada en una ciudad. Para cada par de hiperwalks h $\in$S y h'$\in$S', concatenamos los walks en h con los walks en h' para obtener el conjunto de hiperwalks S 0 S' que satisfacen hwe PARADA cuando CIUDAD dirección HOTEL + PARADA cuando CIUDAD dirección RESTAURANT.

Las operaciones unión(U); intersección($\cap$) y diferencia(|) se definen normalmente entre dos conjuntos de hiperwalks que satisfacen el mismo hwe.

3.6 CONCLUSIONES ACERCA DEL MODELO

De varios modelos de consultas que se han propuesto para solucionar el gran problema de la desorientación (como por ejemplo [*ICHI*93] y [*TAM*93]), consideramos que este es uno de los más importantes debido a la gran poder expresividad que se obtiene al formular las consultas en el lenguaje SQL-like presentado a lo largo de este capítulo. Consideramos de gran importancia poder definir links recursivos para formular consultas sobre esquemas donde hay una estructura de datos bien definida que se repite a lo largo del mismo.

Con el objetivo de obtener un modelo más flexible, propongo hacer una extensión al modelo GRAM. Esta extensión consiste en estudiar como se modifican los elementos existentes en este modelo y agregar los elementos necesarios para integrar el concepto de herencia al mismo. Con la integración de este concepto, obtenemos un modelo más flexible al momento de diseñar el esquema de la aplicación, ya que contamos con las ventajas del diseño orientado a objetos[*REBE*90]. También obtenemos un modelo más flexible al momento de hacer consultas, ya que con la herencia podemos expresar consultas más abstractas o generales, cuando necesitamos que éstas lo sean. En el capítulo siguiente se detalla esta

extensión y al final del mismo hago una conclusión donde podemos observar las ventajas de este modelo y cómo resolvemos el problema de búsquedas y consultas enunciado en el capítulo 1.

CAPITULO 4

EXTENSION AL MODELO PARA INCLUIR HERENCIA

CONTENIDO

En este capítulo hacemos una extensión al modelo presentado en el capítulo anterior. En la primer sección hacemos una introducción al concepto de herencia. En la segunda sección estudiamos la integración de herencia al nivel del esquema de la aplicación. En la tercer y quinta sección vemos como influye la herencia al momento de hacer consultas. En la cuarta sección estudiamos como se afecta el concepto de recursión al integrar herencia al modelo. En la sexta sección integramos el concepto de agregación al modelo. En la séptima y última sección hacemos una conclusión de nuestro modelo detallando las características principales del mismo y cómo sirven para solucionar el problema de búsquedas y consultas enunciado en el capítulo 1.

4.1 INTRODUCCION

En el capítulo anterior mencionamos que los objetos del dominio de una aplicación pueden ser agrupados formando tipos o clases. Aquí agrupamos todos los objetos que tienen iguales características (atributos). Una vez que tenemos todas las clases de la aplicación, podemos observar (como otra etapa del análisis orientado a objetos o entidades) que clases guardan características en común. Ejemplo: supongamos que estamos analizando el dominio de una facultad. En este tendremos , entre otras, las clases ALUMNO, DOCENTE, NO DOCENTE y CATEDRA. Acá podemos observar que tanto ALUMNO como DOCENTE y NO-DOCENTE tienen características en común, ya que todas estas clases se refieren a personas. Entonces podemos crear una clase abstracta PERSONA la cual tendrá los atributos dni, nombre y apellido, dirección, y todos los datos concernientes a una persona independientemente de la función que cumpla en la facultad. A su vez podemos observar que DOCENTE y NO-DOCENTE son personas pagas en la facultad, mientras que ALUMNO es una persona que no es paga en la facultad (como alumno) . Entonces podemos hacer otra abstracción y crear las clases PAGOS y NO-PAGOS, donde DOCENTE y NO-DOCENTE son subclases de PAGOS y ALUMNO es subclase de NO-PAGOS. En la clase PAGOS tendríamos por ejemplo el atributo salario.
No es la intención de este informe dar una clase de cómo se obtiene una factorización correcta de las clases del dominio (para esto ver [*REBE*90]) de una aplicación. Lo que queremos hacer es introducir el concepto de herencia a nuestro modelo. Para ello habrá dos niveles o momentos de integración : a nivel de esquema y al momento de hacer las consultas.

4.2 INTEGRACION A NIVEL DE ESQUEMA

Para obtener esta integración en el esquema debemos contar con un constructor que nos permita definir jerarquías de tipos o clases. Para ello contaremos con un tipo de link predefinido llamado es-un. Este tipo de link nos servirá para establecer esta relación especial entre 2 tipos o clases y formar jerarquías entre las mismas. Donde la clase raíz o más alta en el árbol de la jerarquía es las más abstracta o general y las hojas de este árbol serán las clases concretas o instanciables.

Diremos (siempre a nivel de esquema) que un nodo que tiene un tipo de link es-un entrando al mismo es un super-tipo. Estos tipos son abstractos, es decir, no tendremos instancias del mismo en el grafo de la base de datos.

Por lo tanto la **definición de esquema** se modifica así :
S = (Ns,Es,Vs,Ms,(T U ST U {es-un})) donde T son los tipos instanciables y ST los super-tipos.
No habrá instancias del tipo de link es-un, ya que este tipo de link sirve para representar una relación más bien 'constructiva' que 'semántica'. Decimos constructiva porque los sub-nodos o sub-tipos heredarán los atributos de sus super-tipos. Se construye así el conjunto de atributos de un sub-tipo como su propio conjunto de atributos más los atributos que hereda de su supertipo. También tendremos herencia de relaciones : un sub-tipo heredara todas las relaciones que tiene su super-tipo, ya sean relaciones que parten del super-tipo o que llegan al mismo.

Existen 2 tipos de herencia:
1) simple : un sub-tipo hereda inmediatamente de un solo supertipo.
2) múltiple: un sub-tipo puede heredar de más de un supertipo directamente.
En nuestro modelo trabajaremos con herencia simple.
Diremos que salvo el tipo de link es-un, los otros serán tipos definidos por la AH.
Para aclarar este concepto, llevaremos el esquema presentado en el capítulo anterior, a otro esquema utilizando el concepto de herencia (sólo dibujaremos la parte de PARADA, el resto del grafo queda igual, ver la figura 20).
Queda claro que sólo tendremos en el grafo de la base de datos instancias de MONUMENTO, CIUDAD o RESTAURANT, pero nunca podemos tener una instancia de un supertipo, como LUGAR, ya que un lugar no define nada en concreto.

4.3 CONSULTAS CON HERENCIA

En esta parte nos dedicaremos a ver como influye el concepto de herencia al momento de hacer consultas.
Los conceptos de walk e hiperwalk no se ven afectados porque estos son caminos a nivel del grafo de la base de datos. En cambio los conceptos de we y hwe deben ser revisados, ya que estos son caminos sobre el esquema de la base de datos, y este último fue afectado con el concepto de super-tipo y la relación es-un.

Walk expressions: si consideramos el we CIUDAD dirección HOTEL, este es un we válido, a pesar que el tipo de link dirección no incide directamente en el tipo de nodo HOTEL. El tipo de link dirección incide directamente sobre el tipo de nodo LUGAR, pero como HOTEL es sub-tipo de LUGAR, HOTEL hereda la relación dirección.
Ahora supongamos que en un we aparece un super-tipo, por ej.: CIUDAD dirección LUGAR. Como es natural pensar que tanto un MONUMENTO como un HOTEL y un RESTAURANT son lugares (son sub-tipos de LUGAR) , podemos inferir que tanto CIUDAD dirección HOTEL, CIUDAD dirección RESTAURANT y CIUDAD dirección MONUMENTO están incluidos en el super-walk-expression CIUDAD dirección LUGAR.
Antes de continuar profundizando en esta extensión del modelo daremos algunas definiciones:

Definición : un super-walk-expression (swe, en adelante) es un we que contiene al menos un super-tipo : Sea w = tn1 tl1 tn2 tl2 .... tnn es un swe si existe al menos un tnj ($j \in [1..n]$) que es un **super-tipo**.

Es natural pensar que un swe no puede contener el tipo de link es-un. No consideramos al tipo de link es-un como un tipo navegable (es_un es una relación constructiva para crear jerarquías de clases o tipos).

Definición: El árbol de herencia de un super-tipo t es el subgrafo del esquema que contiene el tipo t con todos sus sub-tipos. Las hojas de este árbol (HOJAS(t)), serán los tipos que se pueden instanciar, tipos concretos.

Definición:
Los we que están incluidos en un swe w están determinados por el conjunto WEI definido de la siguiente manera:

WEI(w) = { tn1 tl1 .. tnj.h .....tnn / tnj.h ∈ HOJAS(tnj) y tnj es un supertipo }

Ejemplo : WEI(CIUDAD dirección LUGAR) = { CIUDAD dirección HOTEL, CIUDAD dirección MONUMENTO, CIUDAD dirección RESTAURANT}

Podríamos pensar que un swe es un esquema de we, ya que no existen instancias directas de un swe en el grafo de la base de datos. Al ver a un swe como un esquema, sus instancias serían todos los we definidos por el conjunto WEI.

A modo informativo diremos que se puede calcular la cantidad de elementos que tiene un conjunto WEI. Sea w un swe donde hay k super-tipos y sea ni = # (HOJAS(ti)) siendo ti el árbol de herencia del super-tipo i-ésimo($\forall i = [1..k]$) $\Rightarrow$ #(WEI(w)) = n1 x n2 x ..... x nk.

Definición: un walk s satisface un swe w $\Leftrightarrow$ s satisface alguno de los we que pertenecen al conjunto WEI(w).

Definición : un super-hiper-walk-expression (shwe, en adelante) es una expresión regular sobre T , que cumple 2 condiciones:

a) Se puede reescribir como una suma de we r = $\sum_{i=1}^{n} ri$, donde existe al menos un we ri (i∈[1..n]) que es un swe.
b) El siguiente conjunto no es vacío (suponemos que ri es swe):

HWEI(r) = { h = r1 + r2 + ...+ ri.k + ... + rn / ri.k ∈ WEI(ri) y h es un hwe}

Análogamente podemos ver a un shwe como un esquema, cuyas instancias son los hwe que están en el conjunto HWEI.

En el capítulo anterior definimos un lenguaje sobre T ,debemos agregar un ítem y agregar una aclaración a esta definición, esto se debe a que pueden haber super-tipos en T.

L(t) = dom(t) si t es un tipo concreto
L(t) = U L(hi) siendo hi ∈ HOJAS(tr) , t es un super-tipo y tr es el árbol de herencia de t
El resto de la definición permanece igual.

Definición : Un hiperwalk h satisface un shwe s $\Leftrightarrow$ h satisface alguno de los hwe que pertenecen al conjunto HWEI(s).

A continuación veremos la herencia como una forma de resumir consultas.
Supongamos que queremos saber cuales son las ciudades que tienen un restaurant , un hotel o un monumento. Si no tuviéramos herencia esta consulta se satisfacería con la unión de las siguientes 3 consultas:

1) select CIUDAD
from CIUDAD dirección RESTAURANT
where true

2) select CIUDAD
from CIUDAD dirección HOTEL
where true

3) select CIUDAD
from CIUDAD dirección MONUMENTO
where true

Nota: cuando queremos seleccionar todos los hiperwalks que están en I(r) , sólo ponemos true en la cláusula where.
En cambio, teniendo incorporado el concepto de herencia al modelo resumimos la unión de estas 3 consultas a sólo la siguiente:

select CIUDAD
from CIUDAD dirección LUGAR
where true
En esta consulta, el supertipo LUGAR se va a instanciar con todos los nodos que sean del tipo HOTEL, RESTAURANT o MONUMENTO que pertenezcan a una ciudad (ya que éstos son los subtipos de LUGAR).

4.4 COMO SE AFECTA EL CONCEPTO DE RECURSION

Otro tema a tener en cuenta al haber incorporado herencia a nuestro modelo es como afecta la recursión a los tipos. Para llegar a una conclusión acerca de este tema primero analizaremos dos ejemplos:
1) Supongamos que queremos representar el contenido de un libro llamado 'El Arte' , el cual habla sobre los artes más importantes como la música, la pintura, etc.. El mismo está dividido en capítulos, y dentro de éstos tenemos secciones. El grafo de la base de datos sería el siguiente:

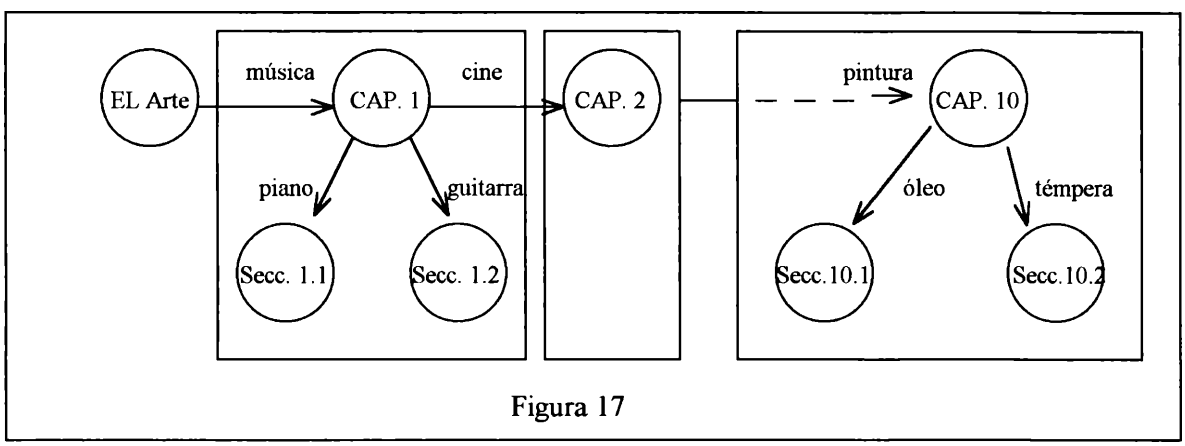


Figura 17

En el grafo anterior podemos observar que tenemos subgrafos (los marcados con un rectángulo) que pertenecen a un mismo esquema de grafo, el esquema :

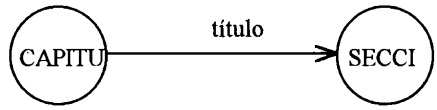


Figura 18

identificando a el tipo CAPITULO como cabeza de estructura. Así nace el concepto de recursión y el nodo que es cabeza de estructura es el nodo sobre el cual se hará recursión. Entones el esquema de un libro podemos decir que es el siguiente :

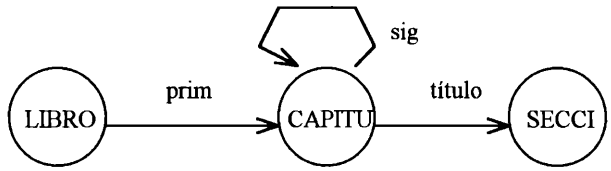


Figura 19

2) Si analizamos el ejemplo de la agencia de turismo identificamos fácilmente que hay una estructura bien definida que se repite a lo largo del grafo de la base de datos. Esa estructura obedece al siguiente esquema:

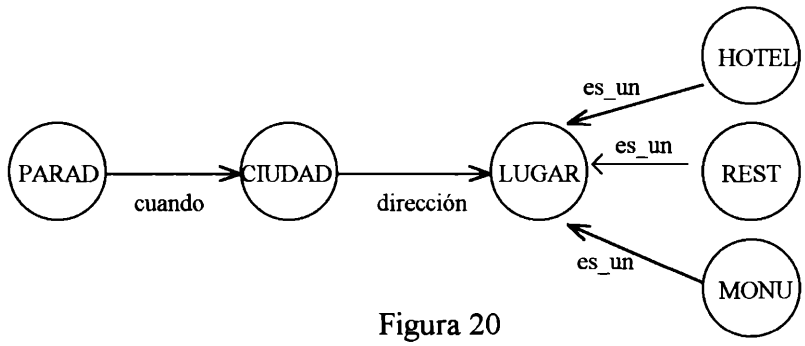


Figura 20

Siendo PARADA la cabeza de estructura.

Si observamos estos ejemplos (y en todos los que hayan estructuras que se repitan) , las cabezas de estructura son entidades bien definidas, **concretas** y **siempre del mismo tipo** (por eso hablamos de repetición). Por lo tanto no tiene sentido tener recursión sobre supertipos, ya que éstos implican varios subtipos instanciables.

4.5 SELECCION Y HERENCIA

Otro factor a tener en cuenta en la inclusión de herencia al modelo , es el criterio de selección cuando tenemos hiperwalks que satisfacen un shwe. Supongamos que queremos ver los lugares que se pueden visitar en Paris. Para esto haríamos la siguiente consulta :
select LUGAR
from CIUDAD dirección LUGAR
where CIUDAD.nombre = 'Paris'

Supongamos también lo siguiente:
LUGAR (año, arquitecto)
RESTAURANT (cant-mozos , etc.)
HOTEL (cant-estrellas, etc.)
MONUMENTO (a-quien, etc.)
donde año es el año en que se construyó ese lugar.

Análogamente a los tipos concretos, podemos poner en la cláusula where que sólo queremos tener aquellos lugares que cumplan cierta característica (esa característica es un atributo del supertipo), por ejemplo:
where : CIUDAD.nombre = 'Paris' y LUGAR.año = 1980

También en la cláusula where se puede hacer referencia a un atributo que no es propio de un tipo concreto sino que es un atributo heredado. Por ejemplo : queremos ver todos los hoteles que se terminaron de construir en 1975 :

select : HOTEL
from : HOTEL
where : HOTEL.año = 1975

4.6 CARDINALIDAD

Incluimos en este modelo el concepto de cardinalidad (también presente en otros modelos como MEYR : modelo de entidades y relaciones). La cardinalidad es un atributo o característica de una relación o link y expresa como se relaciona el tipo origen con el tipo destino de un link. En nuestro modelo tendremos dos tipo de **cardinalidad**:
a) unaria : significa que el tipo origen podrá estar relacionado con a lo sumo 1 nodo del tipo destino. Por ejemplo : el tipo de link cuando tendrá cardinalidad unaria ya que un nodo del tipo PARADA puede estar relacionado con a lo sumo un nodo del tipo CIUDAD (una parada sólo tiene una ciudad a visitar).
b) n-aria : significa que el tipo origen podrá estar relacionado con una cantidad arbitraria de nodos del tipo destino. Un ejemplo de esto es el tipo de link dirección (una ciudad puede tener varios lugares para visitar).

Para incluir este concepto en el esquema agregamos una función al mismo que determina para cada tipo de link instanciable, si es unario o n-ario: Cs: Te $\rightarrow$ {unario, n-ario}.

Hasta el momento tenemos que el **esquema** de la aplicación hipermedial debe ser de la siguiente forma : S =(Ns,Es,Js,Ms,T U ST U {es_un},Cs).

4.7 RELACIONES DE AGREGACION

La inclusión de este tipo de link al esquema nos permite definir agregados de datos en nuestro domino. En otras palabras esto significa poder expresar que una entidad menor forma parte de otra entidad mayor. Veamos un ejemplo:

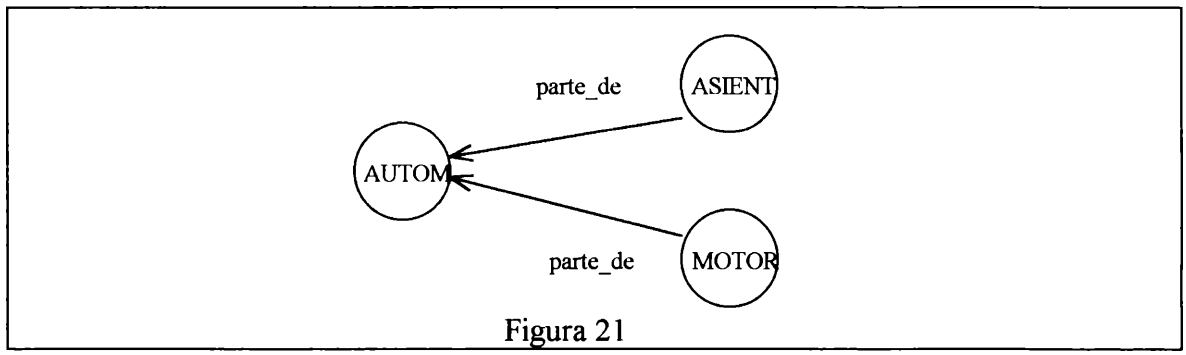


Figura 21

Acá estamos expresando que un asiento es parte de un automóvil (lo mismo sucede con el motor). Ahora bien, observando este sencillo ejemplo podemos deducir que el tipo de link parte_de no cumple con la propiedad de uniformidad de tipos. Esto es muy lógico y natural, ya que esta relación sirve para expresar como están formadas ciertas entidades, y al estar estas entidades formadas por entidades de distinto tipo (en el ejemplo: automóvil está formado por asiento y motor, estos dos últimos tipos son distintos), es lógico que la relación parte_de no sea una relación homogénea. Incluyendo este concepto en el **esquema**, el mismo quedaría conformado finalmente de la siguiente manera:

S=(Ns,Es,Js,Ms,T U ST U { es_un , parte_de },Cs).

4.7.1 RELACIÓN PARTE_DE

Quizás, lo que más análisis requiera al incluir el concepto de agregación, es cuando nos encontramos con el grafo de la base de datos en sí, y la pregunta sería la siguiente: debería ser un tipo navegable el tipo de link parte_de ?, o lo que es lo mismo: tiene sentido tener instancias del tipo parte_de ?.

Podríamos decir que tal vez en algunas aplicaciones sería útil poder consultar los tipos que forman parte de un tipo mayor. Pero a nivel de instancia la pregunta sería la siguiente: Qué label o identificación llevarían estos arcos que unen una instancia mayor con las instancias que son parte de ella?. Veamos un ejemplo (esta instancia corresponde al esquema visto anteriormente):

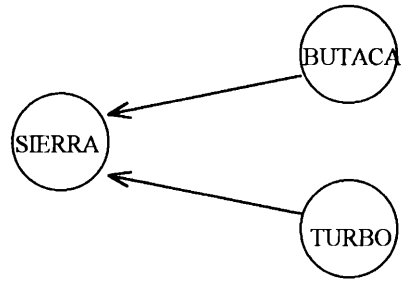
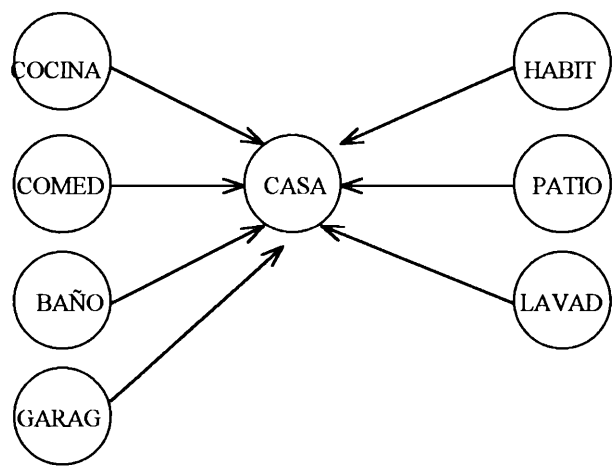


Figura 22

Intuitivamente sabemos que los arcos corresponden al tipo de link parte_de, pero como la computadora por ahora no es intuitiva debemos hallar una forma de distinguir estos arcos. Si por un momento pensamos en el tipo de link dirección (visto en ejemplos anteriores), vemos que este tipo de link tiene atributos como nombre_calle, número, cod. postal. Luego el valor de alguno de estos atributos lo pondremos como label. Sin embargo, la relación parte_de no posee atributos que surgan con naturalidad.

Si el autor del esquema de la base de datos se le ocurre que esta relación posee atributos, éstos serán designados por el mismo. En caso contrario, se me ocurre una solución que podría funcionar en todas aquellas situaciones en las cuales el autor no encuentra forma de identificar o distinguir las instancias de este tipo de link. La solución es sencilla y se basa en dar un solo atributo a este tipo de relación. Por ahora no nos preocuparemos por el nombre de este atributo. Lo que sí importa es que el contenido de este atributo nos va a decir como se comporta o sitúa la sub_entidad que conecta este link con respecto a las otras sub_entidades que son parte de la entidad mayor. Aclaremos esto con un ejemplo. El esquema sería el siguiente:



Aclaración : todos los labels de las flechas son : parte_de.

Figura 23

Ahora bien, si nosotros hacemos una clasificación por importancia, tamaño, costo o el concepto por el cual quisiéramos clasificar, podemos tener un orden entre estos componentes de la casa. Por ejemplo: si nos basáramos en orden de importancia para poder vivir dentro de una casa tendríamos quizás el siguiente orden: habitación, baño, cocina, comedor, lavadero, garaje, patio. Entonces, el grafo de la base de datos puede ser el siguiente:

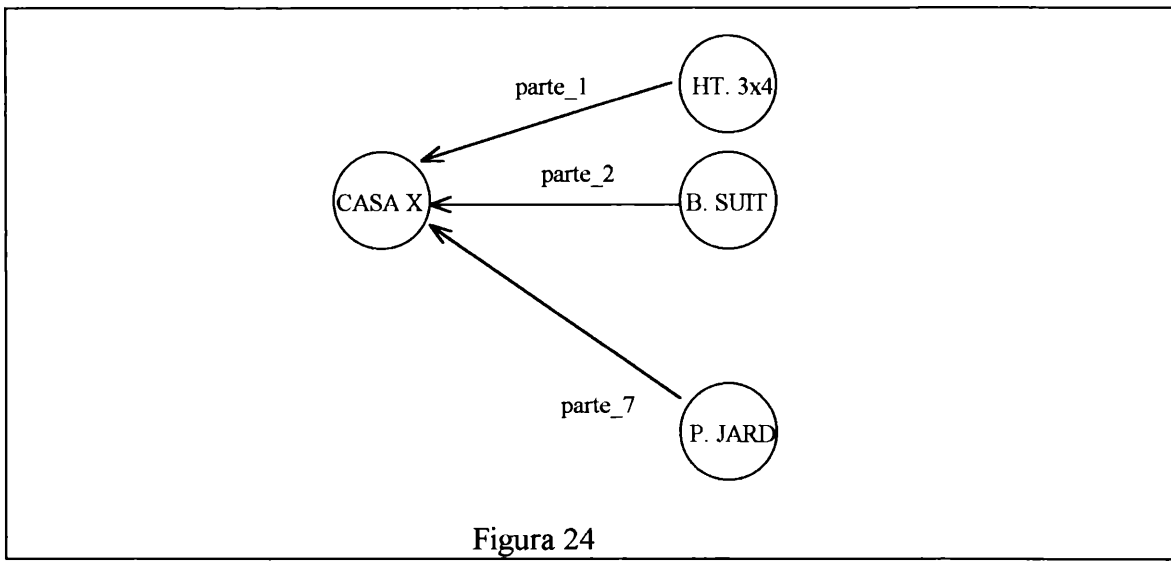


Figura 24

donde parte_1 nos dice que este link une la entidad mayor con su componente primera en una clasificación de acuerdo , en este ejemplo, a su importancia.

4.8 CONCLUSIONES ACERCA DE NUESTRO MODELO

En conclusión podemos decir que estamos en presencia de un modelo muy flexible que permite consultar aplicaciones hipermediales. Estas son las características que tiene el modelo que justifican lo dicho anteriormente:

1. **Tipos de links** : verdaderas entidades . Este modelo ofrece la misma flexibilidad para especificar un tipo de nodo o un tipo de link , ya que un tipo de link puede tener atributos , que es lo que principalmente forma a una entidad como tal. Este aspecto del modelo, complementa a los sistemas de hipermedia que no permitan especificar a los tipos de links como verdaderas entidades. Esto también ofrece la posibilidad de especificar la condición de selección de la consulta en función de los atributos de los tipos de links, no solamente en función de los atributos de los tipos de nodos.

2. **Tipos sinónimos** : este concepto refleja la operación de renombre del Álgebra de hiperwalks. Este concepto posibilita hacer consultas que, sin ellas, nos costaría gran cantidad de tiempo llegar a la información que estamos buscando. En la sección 6.2.3 hay un ejemplo de consulta que usa un tipo sinónimo.

3. **Tipos de links recursivos** : esta es una herramienta muy poderosa, ya que nos permite realizar consultas "potencialmente infinitas" en un espacio finito. Para tener una explicación más detallada de este concepto referirse a la sección 3.4.

4. **Herencia** : este concepto le da mucha flexibilidad al modelo. Podemos mencionar 2 ventajas al especificar un modelo usando herencia: una de ellas es al momento del diseño del esquema , ya que contamos con herencia (de atributos y relaciones o links) y otras ventajas derivadas del diseño orientado a objetos [*REBE*90].
La otra ventaja es al momento de hacer consultas. Incluyendo herencia (a través de tipos abstractos) en las consultas logramos que estas sean más generales cuando necesitamos que lo sean.

5. **Interface optativa** : el servidor que construiremos para reflejar este modelo ofrecerá al usuario la posibilidad de usar la interface que por default tiene incorporado el servidor para hacer consultas o el usuario podrá usar la interface que el desee.

6.**Servidor heterogéneo** : cualquier aplicación (con esto queremos decir que la aplicación puede estar escrita en cualquier lenguaje o desarrollada con cualquier sistema de hipermedia) que corra bajo Windows podrá utilizar al servidor. Esto es posible ya que el servidor exporta 8 funciones atómicas que son suficientes para comunicar el esquema (sInitApp, sAddTnode, sAddTlink, SAddSinon), los datos de la aplicación (sAddNode,sAddLink) y también para hacer una consulta (sQuery y sQueryWin). Las facilidades que la aplicación tiene para mostrar y procesar la respuesta a esa consulta ya dependen del sistema de hipermedia con que se desarrolló la aplicación. Debemos mencionar que hay sistemas de hipermedia que son más flexibles que otros, y esto influirá directamente sobre el proceso y muestra de la respuesta (devuelta por el servidor) a la consulta.

7. **Servidor Multitarea** : el servidor puede ofrecer sus servicios a más de una aplicación a la vez. Esto nos da la posibilidad de relacionar aplicaciones mediante el servidor. En [*HALL*93], [*DAVI*93] y [*HALL*94] se estudia esto en profundidad.

Al final del capítulo 1 mencionamos 7 ítems que deberían cumplirse en las AH. Veamos como con el modelo que hemos creado , aportamos a las AH con herramientas para atacar los problemas que consideramos más importantes:
Con respecto a **búsquedas y consultas** hemos creado un servidor de consultas basado en un lenguaje SQL-like que las aplicaciones podrán utilizar. Al ser SQL-like mantenemos la flexibilidad y expresividad del lenguaje SQL. Este modelo a su vez maneja **composiciones** al ofrecer la posibilidad de obtener estructuras (walks) como resultado de las consultas y no simplemente nodos. Esta estructura se maneja como una verdadera entidad que el usuario podrá navegar. Relacionado a la composición tenemos las **estructuras virtuales**. Podemos afirmar que el walk (elemento en que se basa el Álgebra de Hiperwalks) es una estructura virtual , ya que nace de una instanciación de un pattern de tipos de nodos y links originado por una consulta (esto se explica con detalle en la sección 1.3 ítem 3:estructuras virtuales). Este walk tiene una secuencia de nodos. Cada nodo tiene links, pero sólo serán activados o visualizados los links que nos llevan al siguiente nodo en el walk. De esta forma el walk es

una estructura virtual, porque esa estructura no está almacenada en la AH (sino que se ha creado a partir de una consulta). El servidor cuenta con un **mecanismo computacional** para poder satisfacer las consultas, basado (en grandes rasgos) en la búsqueda, selección y proyección de patterns formados por tipos de nodos y links.
Como se verá más adelante el servidor que hemos construido da la posibilidad al usuario de usar la interface provista por este último o el usuario podrá usar la interface que le plazca para entrar la consulta y ver la respuesta. De esta forma existe cierto de grado de **adaptabilidad** en el servidor, ya que el usuario puede usar la interface que más se le adapte a sus gustos y necesidades.

CAPITULO 5

IMPLEMENTACION DEL SERVIDOR DE CONSULTAS

CONTENIDO

En este capítulo describimos la implementación del servidor de consultas. En la primer sección detallamos que hay que agregarle a una aplicación para que pueda usar al servidor. En la segunda sección describimos como se usa el servidor. En la tercer sección explicamos cada una de las funciones que ofrece el servidor de consultas. En la cuarta y última sección describimos el manejo de errores.

5.1 QUE HAY QUE AGREGARLE A UNA AH PARA USAR EL SERVIDOR

La respuesta es que muy poco (muy poco de código, eso es claro). Sólo hay que agregar código que servirá para lo siguiente:

a- Comunicar al servidor el grafo del esquema y datos.

b- Hay 2 modalidades de usar el servidor. Una de ellas es usar la interface que provee el mismo para entrar la consulta y ver la respuesta. Si se usa esta modalidad hay que agregar un mecanismo (key_stroke o vía menú) para invocar al servidor y ver esa ventana. La otra forma es no usar la interface del servidor. En este caso la aplicación proveerá su propia interface para que el usuario ingrese la consulta y vea la respuesta.

c- Mostrar los nodos que forman parte del camino que el usuario va a navegar (este camino es el que el usuario seleccionó en la respuesta a la consulta, el servidor se lo devuelve a la aplicación para que el usuario lo pueda navegar). Cuando el usuario navega la respuesta (que es un camino), es importante que el usuario no clickee sobre botones que llevan a nodos que no forman parte de la misma. Para ello, es necesario implementar algún mecanismo que evite esa situación. Esto quizás implique tener que cambiar los scripts de los botones (links) en tiempo de ejecución (algo que sólo los manejadores de hipermedia más flexibles permiten hacer). En la implementación optamos por deshabilitar los botones que no llevan al siguiente nodo en la respuesta, pero también podría ser que se muestre un mensaje como "camino temporalmente intransitable" cuando el usuario clickee sobre algún botón que nos haga salir del camino respuesta. Cuando el usuario no quiera navegar más este camino, debe tener la posibilidad de ir a la ventana que inicialmente mostró la repuesta a la consulta (esto se puede hacer vía un comando de un menú). Aquí el usuario podrá elegir otro camino para navegar, ingresar otra consulta o cerrar la ventana.

d- En algún momento el usuario deseará terminar de navegar una respuesta. Aquí el usuario puede optar entre ir al nodo en que estaba al momento de hacer la consulta o podrá ir a algún nodo que pertenece a la respuesta que está navegando. En este último caso el usuario navega el nodo pero ya como cualquier otro nodo que pertenece al hipermedia. Entonces también se necesita de un mecanismo para realizar las 2 opciones enunciadas anteriormente, que podrían implementarse a través de 2 comandos dentro del menú que usamos en b-.

Ahora bien, hay sistemas o generadores de hipertexto/hipermedia que permiten codificar el comportamiento en común a varias aplicaciones (generadas con ese sistema) en un solo lugar. Un ejemplo de ello es ToolBooK. Si observamos el código que hay que agregarle a un aplicación podemos ver que sólo el primer ítem depende de cada aplicación en particular. Entonces teniendo en cuenta esto último , pienso que va a ser de suma utilidad proveer un módulo (clasificado como libro de sistema en el vocabulario ToolBook) que se encargará de proveer todo el comportamiento en común (ítems b,c, y d) que tendrán los libros que quieran interactuar con el servidor. Por lo tanto lo único que le queda por codificar al autor de cada libro (porque son cosas que dependen de cada aplicación en particular) que quiera comunicarse con el servidor es el primer ítem enunciado anteriormente, reduciendo

considerablemente la sobrecarga de código que hay que agregarle a una aplicación para que pueda usar el servidor.
Si vemos gráficamente lo dicho anteriormente:

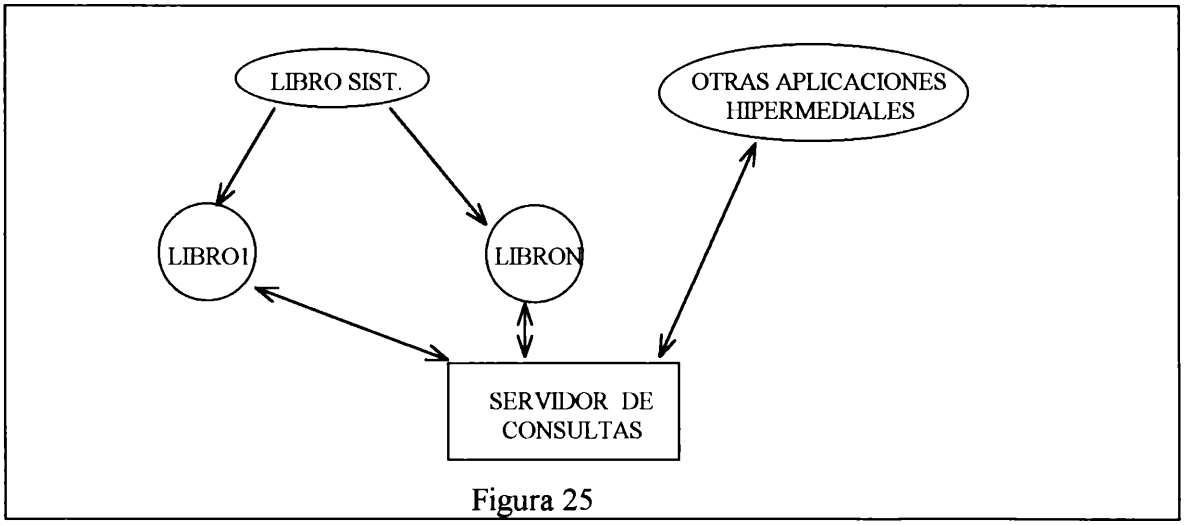


Figura 25

Anteriormente dijimos que hay que agregar un poco de código a las aplicaciones que quieran interactuar con el servidor, por lo tanto es obvio que estas aplicaciones no deben estar "cerradas" totalmente, porque no habrá posibilidad de embeberle este código (hoy en día son pocas las aplicaciones buenas que vienen totalmente cerradas, casi siempre el autor provee algunos fuentes como para custom_izar la aplicación).
Es **importante aclarar** que no es necesario un módulo que contenga comportamiento en común (como en el caso de ToolBook) por cada lenguaje en el cual estén escritas las aplicaciones, simplemente proveo un módulo (con comportamiento común) para ahorrar líneas de código a los autores de las aplicaciones ToolBook.

5.2 COMO SE USA

Para que una aplicación pueda usar el servidor, deberá comunicarle su esquema y datos. Luego de esto podrá hacer todas las consultas que desee y finalmente deberá terminar la sesión con el servidor.
A continuación detallaremos como se hacen cada uno de estos pasos.

5.2.1 COMUNICACION DE DATOS Y ESQUEMA

Para esto el servidor cuenta con 6 funciones que asisten esta tarea. Las mismas son : **sInitApp, sAddTnode, sAddTlink, sAddNode, sAddSinon, sAddLink** (en la sección 5.3 se encuentra una explicación detallada de cada una de ellas) las cuales se encuentran en la DLL con nombre server.dll.

En el caso de aplicaciones ToolBook, deberán agregar la línea : **set sysbook to "c:\..\sistema.tbk"** (los 2 puntos se reemplazan por el path donde se instale el archivo) en el handler **enterBook** (esto es para heredar todo el comportamiento definido en el libro del sistema del cual se habló en la sección 5.1 y se ve en la figura 25). También deberán

codificar un handler en el libro llamado **iniciar** que **sólo** contendrá todas las llamadas a estas funciones que sean necesarias para comunicar el esquema y datos de la aplicación.

5.2.2 COMO SE HACE UNA CONSULTA

En la sección 5.1 dijimos que hay 2 modalidades de usar el servidor. Usando su interface o no. En caso de no usar su interface se podrá usar la función **sQuery** (explicada con detalle en la sección 5.3). Si se usa la interface que provee el servidor se usará la función **sQueryWin** (explicada con detalle en la sección 5.3). Ambas funciones se encuentran en la DLL server.dll.

A continuación detallamos como se efectúa una sesión de consulta para aplicaciones ToolBook que desean utilizar la interface provista por el servidor:

Existirá un menú con el nombre **Consultas**. A través del comando **Consultar** comenzará una sesión de consulta. Se abrirá una ventana donde se ingresa la consulta deseada (para ver esta ventana ir a la sección 5.3). La respuesta a la consulta se muestra en el ListBox y consiste en un conjunto de walks que satisfacieron la consulta. En este momento el usuario podrá seleccionar alguno de estos caminos para poder navegarlo. Cabe aclarar que los nodos que forman parte del camino sólo tendrán activos los botones (links) que le permitan al usuario pasar al próximo nodo del walk (los otros botones estarán desactivados para que el usuario no tome un camino que no pertenezca a la respuesta de la consulta, en otras palabras, para que el usuario "no se vaya por las ramas"). Las otras opciones del menú son Página_actual, Página_anterior y Respuesta. **Página_anterior** sale del camino que se está navegando y lleva al usuario al mismo nodo que estaba al momento de hacer la consulta, finalizando así una sesión de consulta. **Respuesta** abre la ventana de consultas pero muestra la última consulta efectuada junto con su respuesta. Aquí el usuario podrá seleccionar otro walk (camino) para ir a navegar, podrá ingresar otra consulta o cerrar esta ventana. La opción **Página_actual** tiene el mismo efecto que Página_anterior pero con la diferencia que el usuario va a la página (nodo) que se encontraba navegando al seleccionar este comando. Luego de Página_anterior/_actual, el usuario se encontrará navegando una página como cualquier otra del hipermedia, es decir, ya no se encuentra más navegando la respuesta.

5.2.3 COMO FINALIZAR UNA SESION CON EL SERVIDOR

Antes de que la aplicación termine, deberá llamar la función **sRelApp** (esta función se detalla en la sección siguiente). Este función está en la DLL server.dll.

En el caso de las aplicaciones ToolBook, deberán agregar la línea **send terminar** en el handler **LeaveBook**.

5.3 DISEÑO DEL SERVIDOR

El servidor es una librería de enlace dinámico (es una DLL implementada en el lenguaje C para el entorno Windows) que exporta 9 funciones. Estas funciones las podemos clasificar en 4 grupos: funciones de comunicación del esquema de la aplicación , funciones de comunicación de los datos de la aplicación, funciones para realizar consultas y el último grupo posee una sola función para terminar la sesión con el servidor. A continuación detallamos cada uno de estos grupos, los parámetros de las funciones se describen usando los tipos implementados en el lenguaje C.

5.3.1 FUNCIONES PARA COMUNICAR EL ESQUEMA Y DATOS DE LA APLICACION

GLOBALHANDLE **sInitApp**(LPSTR nombre,BYTE ntnodes,BYTE ntlinks,WORD nnodes,BYTE ntnodes_h)

Esta función es la primera que se debe llamar. Ya que sirve para informar al servidor acerca de la envergadura de la aplicación hipermedial a conectar.

requiere:
ntnodes : cantidad total de tipos de nodos.

ntondes_h : aquí se indica de la totalidad de tipos de nodos, cuántos son jerárquicos. El resto de los tipos serán concretos o instanciables.

ntlinks : cantidad de tipos de links.

nnodes: cantidad de nodos.

retorna: una estructura de 1 WORD . El byte menos significativo de este word es un código de error o 0 si no hubo error. En el byte más significativo tenemos la identificación de aplicación o 0 si hubo error.

Nota: **la identificación de la aplicación debe usarse en el parámetro app_id del resto de las funciones.**

BYTE **sAddTnode**(BYTE app_id,LPSTR nombre,LPSTR nom_attrib,BYTE attrib_dis LPSTR tattrib,char mode,LPSTR sup_tnode)

Esta función sirve para informar al servidor acerca de un tipo de nodo.

requiere:
nombre: nombre del tipo que se está agregando.

nom_attrib: nombres de los atributos que tiene el tipo. Los nombres se deben ingresar separados por un carácter blanco. Al menos se deberá ingresar un nombre, que será el nombre del atributo distinguido o clave.

attrib_dis: aquí se indica un nro. que es el índice dentro de la lista de nombres de atributos que será el atributo distinguido. El valor de este atributo lo usará el servidor para indicar que el nodo que corresponde a ese valor ha sido seleccionado en una consulta. Aclaremos con un ejemplo :

select: hotel
from: hotel
where:

Para que el servidor diga cuáles nodos fueron seleccionados, mostrará el valor del atributo distinguido de los nodos de tipo hotel (que son los que satisfacen la consulta).

tattrib: acá se indican los tipos de los atributos mencionados en el parámetro nom_attrib. Este string tendrá tantos caracteres como nombres de atributos hayamos definido. Cada carácter define un tipo para el atributo. El carácter podrá ser c (si el atributo es carácter), s (si es string) o i(si es integer).

mode : modo del tipo de nodo .Se usará c (o ascii 99) si el tipo es concreto , este tipo se podrá instanciar. r (o ascii 114) si es un tipo raíz, este será un tipo jerárquico no instanciable. O s(o ascii 115) si es subtipo jerárquico, este tipo se podrá instanciar sólo si es una hoja, es decir, si no tiene subtipos.

sup_tnode: si se está agregando un tipo del modo s, este parámetro es el nombre del tipo que es el ancestro inmediato, o este parámetro es null en caso contrario.

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

BYTE **sAddTlink**(BYTE app_id,LPSTR nombre,LPSTR nom_attrib,BYTE attrib_dis,LPSTR tnodeOr,LPSTR tnodeDe,LPSTR tatrib)
Esta función sirve para informar al servidor acerca de un tipo de link.

requiere:
nombre,nom_attrib,attrib_dis,tattrib : son análogos a la función sAddTnode.
tnodeOr y tnodeDe : definen el tipo de nodo origen y destino que unirá el tipo de link.

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

BYTE **sAddSinon**(BYTE app_id,LPSTR tnode,LPSTR nombre_sinon)
Esta función refleja la operación RENOMRE del Álgebra de Hiperwalks. Renombrando un tipo de nodo se obtiene un sinónimo del mismo.

requiere:
tnode: tipo del nodo del cual que se quiere crear un sinónimo.
nombre_sinon: nombre del tipo que es sinónimo.

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

5.3.2 FUNCIONES PARA LA COMUNICACION DE LOS DATOS DE LA APLICACION.

BYTE **sAddNode**(BYTE app_id,LPSTR tnode,LPSTR attrib,WORD id_obj)
Esta función sirve para informar al servidor acerca de un nodo.

requiere:
tnode: nombre del tipo de nodo que se quiere instanciar.

attrib : string con los valores de los atributos. Si el tipo de nodo es concreto se deben indicar tantos valores como atributos fueron definidos en el tipo. Si el tipo es jerárquico (y debe ser una hoja) se especificarán estos valores comenzando por los atributos definidos en la hoja, luego con los definidos en su ancestro inmediato y así sucesivamente hasta llegar a la raíz. Los valores de los atributos deben estar separados por un carácter blanco.

id_obj : esta es la identificación que tiene el nodo dentro de la aplicación. Esta identificación es un número de nodo que el sistema de hipermedia (que se usó para construir la aplicación) le asignó al nodo.
Para aplicaciones ToolBook este valor es la propiedad ID de la página.
En caso de que el sistema de hipermedia asigne un valor que no es un número, el usuario deberá crear una tabla que contendrá 2 ítems por entrada. Un ítem es el valor que asignó el sistema de hipermedia al nodo y otro es el valor id_obj. De esta forma cuando el servidor devuelva un número de nodo (para más detalles ver el grupo siguiente de funciones) ,se buscará en esta tabla este valor y se obtendrá su ítem asociado, que es la identificación que el sistema de hipermedia le asignó al nodo, pudiendo de esta forma reconocer el siste ma de hipermedia al nodo que formó parte de la respuesta y mostrarlo para que el usuario lo navegue.

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

BYTE **sAddLink**(BYTE app_id,LPSTR tlink,LPSTR nodeOr,LPSTR tori,LPSTR nodeDe,LPSTR tdes,WORD id_obj, LPSTR attrib)
Esta función sirve para informar al servidor acerca de un link.

requiere:
nodeOr y nodeDe : valor del atributo distinguido del nodo origen y destino que une el link.

tori y tdes : tipos de los nodos ingresados en los parámetros nodeOr y nodeDe.

id_obj: es análogo al de la función sAddNode.
En caso de las aplicaciones ToolBook este valor es la propiedad ID del botón que representa al link.

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

5.3.3 FUNCIONES PARA HACER UNA CONSULTA

Como ya mencionamos en la sección 4.2.2 tenemos 2 funciones para hacer una consulta:

GLOBALHANDLE **sQuery** (BYTE app_id,LPSTR select,LPSTR from,LPSTR where, BYTE we_proy)

requiere:
from : este es el conjunto inicial de búsqueda. La sintaxis para especificar este valor es la siguiente:

from::= we1 [+we2+.....+wen]
wei ::= tn1 [tl1 [)] tn2.....tnn] para i=1..n
tni ∈ tipos de nodos de la aplicación , i=1..n
tli ∈ tipos de links de la aplicación, i=1..n-1

donde cada walk_expression (wei, i ∈ [2..n-1]) comparte al menos 1 tipo de nodo (concreto o abstracto) con los walk_expressions adyacentes. El tipo de nodo que un we comparte con el we predecesor puede que no sea igual al tipo de nodo que comparte con el we sucesor.
Para especificar un tipo de link recursivo se le debe agregar el símbolo) detrás del tipo de link (esto siempre y cuando el tipo de link haya sido definido como recursivo en la función sAddTlink , esto significa haber definido el mismo tipo de nodo origen y destino para el tipo de link).

we_proy : es un número que indica sobre cual we del from se va a realizar la proyección.

select : acá se especifica la proyección de la consulta. Su sintaxis es la siguiente:

select ::= tn1 tl1 ..... tnn tni ∈ tipos de nodo de la aplicación , i=1..n
tli ∈ tipos de links de la aplicación, i=1..n

Si estamos proyectando sobre un we que contiene recursión (esto significa que tiene el símbolo) detrás de un tipo de link) y el select incluye el link recursivo , entonces se tomará el link como recursivo (y no hace falta especificar el símbolo) detrás del link recursivo).
Ejemplo (para ver el esquema sobre el cual se hace la consulta, ver la sección 6.2.3) :

select : CIUDAD sig CIUDAD
from : PAIS tiene CIUDAD sig) CIUDAD dir HOTEL
where :

El we del from tiene recursión y el select está incluido en la parte de la recursión (notar que no hace falta poner el signo) en la cláusula select). Por lo tanto, la estructura de los walks que satisfagan la consulta podrá será : CIUDAD , CIUDAD sig CIUDAD, CIUDAD sig CIUDAD sig CIUDAD y así sucesivamente.
Si el select fuera CIUDAD dir HOTEL, no incluye recursión y la estructura de los walks que satisfacen la consulta será CIUDAD dir HOTEL (igual que el select).

where: es una expresión totalmente encerrada entre paréntesis que especifica la condición de selección de la consulta. Totalmente encerrada entre paréntesis significa que se debe encerrar entre paréntesis cada expresión que contenga un conectivo lógico. Los conectivos lógicos permitidos son el & (que es el and) y el | (que es el or).
Para hacer referencia a un atributo se lo hace con el formato tipo.atributo[*we*], donde we es el número de we que nos estamos refiriendo dentro del campo from.
Una condición de selección puede ser de la forma : tipo.attrib[*we*] *con* tipo.attrib[*we*] o puede ser de la forma tipo.attrib[*we*] *con* valor. Donde *con* puede ser <,≤ , <> , > , >=,=.

Si nos estamos refiriendo a un we que tiene más de una ocurrencia del tipo que está en la condición de selección , basta con que el valor de al menos 1 de esas ocurrencias cumpla con la condición como para que esa condición sea verdadera para el walk que se está analizando.
Ejemplo: PARADA en PARADA, condición : (PARADA.nombre[we]=Parada1) y el walk: Parada1 bus Parada2. La condición es verdadera para el walk ya que al menos 1 instancia del tipo PARADA es Parada1.

Ejemplos de expresiones bien formadas son:

(tipo.atributo[*we*] ≤ valor)
(tipo.atributo[*we*]= tipo.atributo[*we*] & tipo.atributo[*we*] <>valor)
(tipo.atributo[*we*] = valor | (tipo.atributo[*we*] > valor &
tipo.atributo[*we*] >=valor))

retorna: un globalhandle a una estructura de 3 bytes con el siguiente formato :

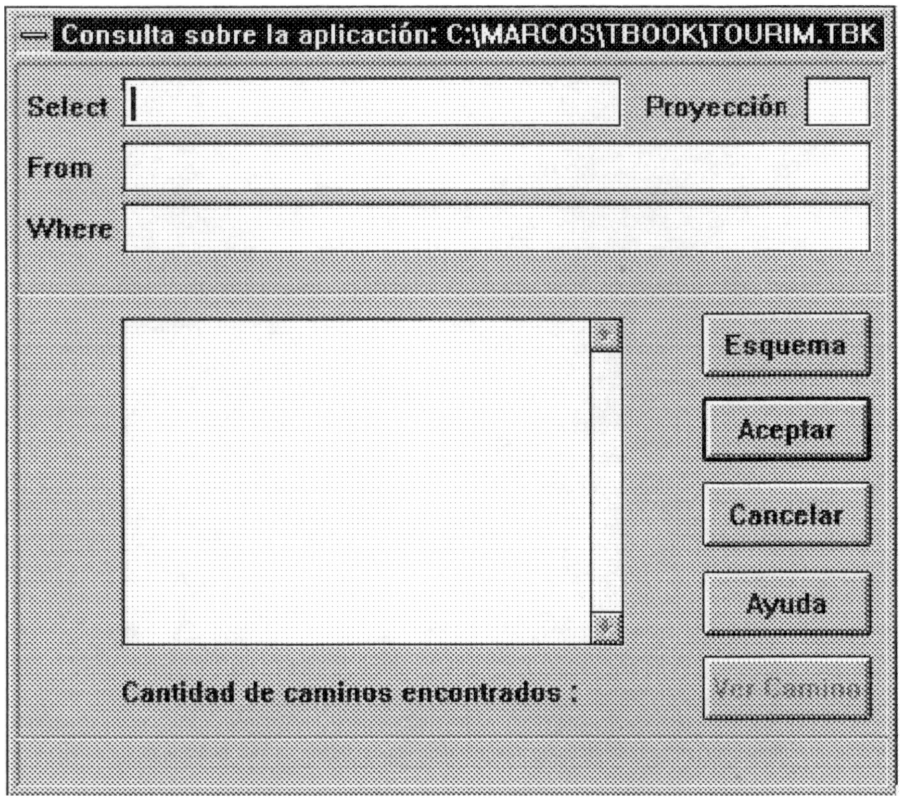
BYTE : que es 0 si no hubo error o distinto de cero si lo hubo.
WORD : Si no hubo error, esta palabra tiene un globalhandle a la siguiente estructura:

* WORD : es la cantidad de walks que satisfacieron la consulta.
WORD : es un globalhandle que contiene las identificaciones de los nodos/links que satisfacieron la consulta. Se tiene un walk seguido de otro. Esta identificación es la que se especificó en la función sAddNode/sAddlink parámetro id_obj.
BYTE : número de elementos que tiene un walk
WORD : es un globalhandle que contiene un string con los atributos distinguidos de cada nodo/link del walk.

Los 2 últimos campos se repiten tantas veces como diga el primer campo *.

La otra función para hacer una consulta (utilizando la interface provista por el servidor) es :
GLOBALHANDLE **sQueryWin**(BYTE app_id, BYTE estado)

requiere:
estado : indica si al abrir la ventana se mostrará la última consulta efectuada y su respuesta (estado = 1) o si se va a efectuar una nueva consulta (estado = 0).
Esta función abre la siguiente ventana:



La parte inferior de la pantalla es sensible a la posición del mouse y mostrará una leyenda de ayuda acerca del campo donde se encuentra el mouse.

En los **campos select, from ,where y proyección** se ingresa la consulta que se quiere realizar con la sintaxis descripta para la función **sQuery**.

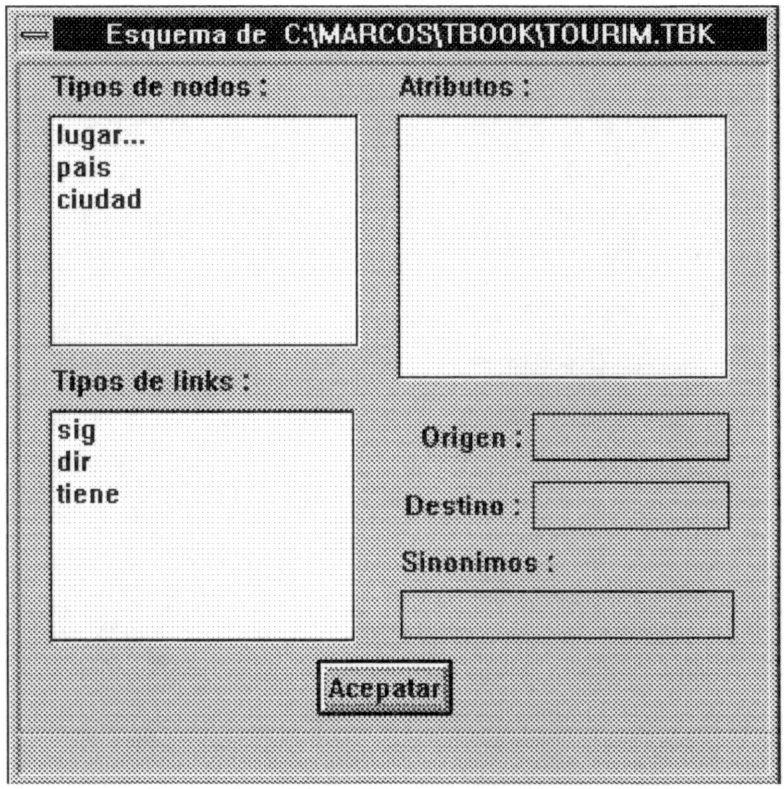
El **botón Aceptar** es sensible al campo que tuvo el último foco de atención. Si el foco estuvo en los campos descriptos arriba, se efectuará la consulta y se mostrará el resultado en el ListBox (área que está en la parte inferior izquierda de la ventana). Si el foco está en la lista, se cerrará la ventana y la función devolverá el camino seleccionado a la aplicación para que ésta permita que el usuario lo navegue (más adelante veremos la estructura en que se devuelve el walk seleccionado).

El **botón Cerrar** cierra la ventana.

El **botón Ayuda** muestra una ventana con la explicación de los campos de la ventana que estamos describiendo.

El **botón Ver camino** se activará sólo cuando alguno de los walks que se muestran en la lista supera el ancho de la misma. Entonces a través de este botón se puede ver en su totalidad el camino seleccionado.

El **botón Esquema** abre la siguiente ventana :



En la parte inferior de la ventana se muestra una leyenda de ayuda que es sensible a la posición del mouse.
En la **lista tipos de nodos** se muestran los tipos de nodos de la aplicación. Los tipos abstractos se mostrarán como tipo_abstr...(en la ventana se ve el tipo lugar) y si uno clickea sobre éste se mostrarán sus subtipos inmediatos en forma indentada para que se sepa que son subtipos del tipo que se clickeó. Si se vuelve a clickear sobre tipo_abstr se realiza el proceso inverso.
En la **lista tipos de links** se muestran los tipos de links de la aplicación.
En la **lista de atributos** se muestran los nombres y tipos de los atributos del tipo de nodo/link seleccionado.
En los **campos Origen** y **Destino** se muestran el tipo de nodo origen y destino del tipo de link seleccionado.
En el **campo Sinónimos** se muestran los sinónimos del tipo de nodo seleccionado (si tuviera).
El **botón Aceptar** cierra la ventana.
retorna:
La función **sQueryWin** retorna 0 si se canceló o cerró la ventana . O retorna un GLOBALHANDLE a la siguiente estructura si se presionó el botón aceptar y había un camino seleccionado:
BYTE : cantidad de elementos (nodos y links) del walk que estaba seleccionado en la ventana.
WORD : identificación del nodo/link que forma el walk. Este número es el que se le pasó a la función sAddNode/sAddLink parámetro id_obj.
Este último campo se repetirá tantas veces como lo indica el campo anterior.

5.3.4 La última función es para terminar una sesión con el servidor:
BYTE **sRelApp**(BYTE app_id)

retorna:
Un número que es 0 si no hubo error u otro valor si lo hubo.

Es importante llamar a esta función antes de terminar la aplicación ya que el servidor en este momento liberará toda la memoria que le destinó a la aplicación. Esta función se encuentra también en la DLL server.dll.

5.4 MANEJO DE ERRORES

Hay dos momentos en que el servidor indicará posibles errores. Uno de ellos es durante la comunicación del esquema y datos. El otro momento es cuando se efectúa una consulta. Todos los errores se presentan de la misma forma, en una ventana con el siguiente formato:

donde n es el número de error y algún_lugar es el nombre de la función donde se originó el error (en el caso de tener un error durante la comunicación del esquema y datos) y en caso de que estemos haciendo una consulta , indica en que campo de la consulta se generó el error (esto en caso de usar la interface que provee el servidor para hacer una consulta).

5.4.1 ERRORES AL MOMENTO DE COMUNICAR EL ESQUEMA Y DATOS

Las funciones son operaciones atómicas. Es decir, si ocurre un error en la función , luego de mostrar el error , el servidor se comporta como si la función no se hubiera ejecutado en su totalidad.

Errores surgidos en la función : **sInitApp**(LPSTR nombre,BYTE ntnodes,BYTE ntlinks,
WORD nnodes,BYTE ntnodes_h)

1	Debe haber por lo menos 2 tipos de nodos
2	Debe haber por lo menos 1 tipo de link
3	Cantidad total de nodos debe superar cantidad de nodos jerárquicos
4	Aplicación existente en el servidor
5	Memoria insuficiente

Errores surgidos en la función **sAddTnode**(BYTE app_id,LPSTR nombre,LPSTR nom_attrib,BYTE attrib_dis LPSTR tattrib,char mode,LPSTR sup_tnode)

1	La aplicación no existe en el servidor
2	La lista de tipos de atributos debe tener por lo menos 1 ítem
3	Hay un tipo de atributo que no es válido
4	Hay un modo que no es válido
5	Supertipo inexistente
6	No hay más capacidad para agregar más tipos de nodos comunes
7	No hay más capacidad para agregar más tipos de nodos jerárquicos
8	Tipo de nodo existente
9	Cantidad de tipos de atributos es distinta de cantidad de nombres de atributos
10	Atributo distinguido no esta entre 1 y cantidad de atributos
11	El nombre del tipo de nodo ya existe como nombre de tipo sinónimo
12	El atributo distinguido se hereda de su supertipo
13	En la lista de nombres de atributos hay un nombre que está repetido
14	Memoria insuficiente

Errores surgidos en la función **sAddLink**(BYTE app_id,LPSTR tlink,LPSTR nodeOr, LPSTR tori, LPSTR nodeDe,LPSTR tdes, LPSTR attrib, WORD id_obj)

1	La aplicación no existe en el servidor
2	No existe el tipo de link
3	Cantidad de valores en la lista de atributos distinta de cantidad de atributos definidos en el tipo
4	Hay un valor de atributo que debería ser carácter y no lo es
5	Hay un valor de atributo que debería ser integer y no lo es
6	El nodo origen no es del tipo especificado por el tipo del link
7	El nodo destino no es del tipo especificado por el tipo del link
8	No existe el nodo origen
9	No existe el nodo destino
10	Memoria insuficiente

Errores surgidos en la función **sAddTlink**(BYTE app_id,LPSTR nombre,LPSTR nom_attrib, BYTE attrib_dis, LPSTR tnodeOr, LPSTR tnodeDe,LPSTR tatrib)

1	La aplicación no existe en el servidor
2	No hay capacidad para agregar más tipos de links
3	Hay un tipo de atributo que no es válido
4	Tipo de nodo origen inexistente
5	Tipo de nodo destino inexistente
6	Tipo de link existente
7	Cantidad de tipos de atributos distinta de cantidad de nombres de atributos
8	El atributo distinguido no está entre 1 y cantidad de atributos
9	En la lista de nombres de atributos hay un nombre que está repetido
10	Memoria insuficiente

Errores surgidos en la función **sAddSinon**(BYTE app_id,LPSTR tnode,LPSTR nombre_sinon)

1	La aplicación no existe en el servidor
2	No existe el tipo de nodo del cual se quiere crear un sinónimo
3	El nombre del sinónimo ya existe como nombre de tipo de nodo
4	El nombre del sinónimo ya existe como nombre de sinónimo de algún tipo de nodo
5	El tipo del cual se quiere crear un sinónimo es jerárquico pero no es hoja
6	Memoria insuficiente

Errores surgidos en la función **sAddNode**(BYTE app_id,LPSTR tnode,LPSTR attrib,WORD id_obj)

1	La aplicación no existe en el servidor
2	No existe el tipo de nodo
3	El tipo de nodo es jerárquico pero no es hoja (no se puede instanciar tipos abstractos)
4	Cantidad de valores de atributos es distinta de cantidad de atributos definidos en el tipo del nodo
5	Hay un valor de atributo que debería ser carácter y no lo es
6	Hay un valor de atributo que debería ser integer y no lo es
7	El tipo que se quiere instanciar no tiene definido el atributo distinguido
8	No hay capacidad para agregar más nodos
9	Memoria insuficiente

Errores surgidos en la función **sRelApp**(BYTE app_id)

1	La aplicación no existe en el servidor

5.4.2 ERRORES SURGIDOS AL MOMENTO DE HACER UNA CONSULTA

Ya sea usando la función **sQuery** (en este caso el error se devuelve en un parámetro) o usando la función **sQueryWin** (el error se muestra en una ventana, ya que con esta función se utiliza la interface que provee el servidor para hacer consultas).

Los números de errores están organizados de la siguiente forma:
error>50 y error <100 : error en el campo from
error>100 y error ≤109 : error en el campo select
error=110 : error en el campo proyección
error>150 : error en el campo where

Errores surgidos en el campo **from**

51	Memoria insuficiente para realizar la consulta
52	No existe un tipo de nodo origen
53	No existe un tipo de nodo destino
54	No existe un tipo de link
55	El tipo de nodo origen no es correcto para un tipo de link
56	El tipo de nodo destino no es correcto para un tipo de link
57	No existe un tipo de nodo
58	Un tipo de link no es recursivo
59	El campo from es vacío
60	El campo from no contiene un hiperwalk_expression
61	Hay un número par de tokens
62	El from termina con carácter + o)
63	El símbolo) no puede ir detrás de un tipo de nodo
64	Hay 2 we iguales que son consecutivos

Errores surgidos en el campo **select**

101	Memoria insuficiente para realizar la consulta
102	No existe un tipo de nodo origen
103	No existe un tipo de nodo destino
104	No existe un tipo de link
105	El tipo de nodo origen no es correcto para un tipo de link
106	El tipo de nodo destino no es correcto para un tipo de link
107	No existe un tipo de nodo
108	Número par de tokens
109	El select está vacío

Errores surgidos en el campo **proyección**

110	Índice de proyección fuera de limites o el select no es proyección del we indicado

Errores surgidos en el campo **where**

151	Hay una expresión que no se puede separar en tipo.atributo
152	Hay una expresión que tiene un tipo de nodo/link inexistente
153	En la expresión tipo.atribb , attrib no es un atributo de tipo
154	No hay una expresión antes de un conectivo lógico & o |
155	Mal anidamiento de paréntesis
156	Más de un conectivo lógico en una expresión encerrada por paréntesis
157	En la expresión tipo.atributo[*n*] , *n* no es un número
158	Falta el carácter) después de la expresión tipo.atributo[*n*]
159	Mal anidamiento de paréntesis , falta el carácter)
160	Expresión inválida
161	Hay un tipo de nodo/link que no está en el we indicado
162	Memoria insuficiente para hacer la consulta
163	No se comparan atributos de igual tipo
164	Hay un valor que debería ser carácter y no lo es
165	Hay un valor que debería ser integer y no lo es
166	Hay un valor aislado
167	No hay expresión antes del carácter)
168	En la expresión tipo.atributo[ ] no se indica el we al que se refiere la condición
169	En la expresión tipo.atributo[*n*] , n no es un número de we válido
170	Hay una condición de selección que no tiene signo de comparación (=, < , >= , etc.)
171	faltan [ ] para indicar un número de we

CAPITULO 6

EJEMPLOS Y CLASIFICACION DE APLICACIONES HIPERMEDIALES

CONTENIDO

En este capítulo contamos con tres secciones. La primera de ellas es una introducción al capítulo. La segunda contiene 3 ejemplos de aplicaciones hipermediales y consultas sobre las mismas utilizando la sintaxis entendida por el servidor. La tercer y última sección contiene una clasificación de aplicaciones según su grado de adaptabilidad para poder interactuar con el servidor de consultas.

6.1 INTRODUCCION

En este informe presentamos tres ejemplos de aplicaciones hipermediales y veremos la utilidad que brinda un servidor de consultas en este tipo de aplicaciones. También haremos una clasificación de las aplicaciones hipermediales en función del grado de su adaptabilidad para interactuar con el servidor.

6.2 EJEMPLOS

6.2.1 PRIMER EJEMPLO

Primero que todo, intenté buscar una aplicación que tuviese una gran cantidad de información. Una de ellas es el CD "Art Gallery" de Microsoft. Debido a que no se puede agregar código a esta aplicación en CD para que pueda usar al servidor, decidí diseñar una aplicación similar a efectos de mostrar las consultas que se podrían hacer en caso de que esta aplicación en CD se le pudiera embeber código. El siguiente paso fue inspeccionar la información contenida en el CD con el propósito de tiparla, es decir, identificar las entidades que estaban presentes en esa información, clasificarlas en clases o tipos, y más tarde ver las relaciones que existían entre ellas.

Esta no es una tarea fácil, ya que el CD "Art Gallery" contiene muchísima información. Como es muy costoso en tiempo (e innecesario para lo que queremos mostrar en este informe) inspeccionar todos y cada uno de los nodos, me dediqué principalmente a identificar los nodos y relaciones que serían de utilidad modelar como para poder ser consultados por un usuario de esta aplicación. En el esquema que se presenta a continuación no se muestran nodos que son índices, ya que ésta no es la información en sí, sino que son nodos que sirven para acceder a los nodos que sí tienen información, y que son los que se han modelado en el esquema. A los tipos que hemos identificado, les hemos asignado atributos, que serán utilizados para establecer condiciones de selección en las consultas. Con respecto a las relaciones hay algo para aclarar. Si por ejemplo tomamos las relaciones que existen entre 2 autores , éstas son muchísimas (por ejemplo hijo, discípulo, ayudante, etc. etc.). Como no es bueno tener un arco por cada tipo de relación que haya (en este caso esta situación llevaría a tener muchísimos arcos innecesariamente entre 2 AUTORES, tendríamos una estructura muy compleja sin necesidad), creamos una relación ficticia (en el CD no existe) que abstrae todas estas relaciones. A esta relación le damos algún nombre : relación_1 que tendrá un atributo llamado tipo con dominio : hijo , discípulo , ayudante ,etc. etc. Una situacion análoga se encuentra entre PAIS y AUTOR.

El esquema es el siguiente:

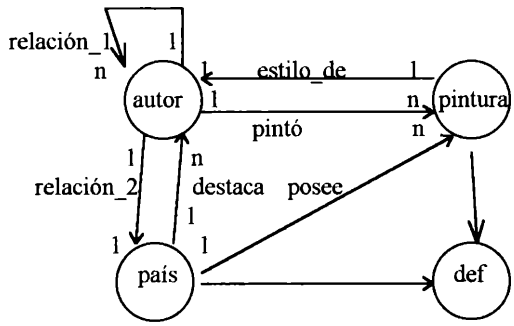


Figura 26

Nota : los arcos que llegan a def no me interesé en ponerles algún tipo porque no los usaré en las consultas. De todos modos podrían tener un tipo sólo a modo de identificación. Idem para los atributos de los arcos estilo_de, pintó, destaca y posee.

Atributos:
AUTOR = { nombre , fecha_nac, foto, etc.}
DEF = { definición }
PINTURA = { título, año_pintó, tamaño, descripción, número , género}
PAIS = { nombre, tiempo, etc.}
RELACION_1 = {tipo} , dom(tipo)= { hijo, discípulo , ayudante , etc.}
RELACION_2 = {tipo} , dom(tipo) = { nació, mudó, murió, etc.}

Veamos las consultas que se podrían hacer (ya con la sintaxis especificada en el capítulo anterior):

1) Discípulos de Leonardo Da Vinci:
select: AUTOR´ proyección:1
from AUTOR relación_1 AUTOR´
where (AUTOR.nombre[1] = Leonardo_Da_Vinci & relación_1.tipo[1] = discípulo)

Esta es una consulta sencilla, pero si no tuviéramos la posibilidad de hacer consultas, tendríamos que ir hasta el nodo "Leonardo Da Vinci" y clickear para ver un discípulo. Para ver otro discípulo debemos volver al nodo "Leonardo Da Vinci" y clickear nuevamente para ver otro discípulo. Esto último lleva mucho tiempo, pero con la posibilidad de hacer una consulta, obtenemos directamente una lista de los discípulos buscados.

2) Discípulos de Leonardo nacidos en Italia:
select : AUTOR´ proyección:1
from : AUTOR relación_1 AUTOR´ relación_2 PAIS
where: ((relación_2.tipo[1] = nació & PAIS.nombre[1] = Italia) & (relación_1.tipo[1] = discípulo & AUTOR.nombre[1]=Leonardo_Da_Vinci))

3) Autores que se mudaron a París y pintaron sobre óleo:
select : AUTOR proyección:1
from : AUTOR relación_2 PAIS + AUTOR pintó PINTURA
where: (relación_2.tipo[1] = mudó & (PAIS.nombre[1] = Francia & PINTURA.género[2] = óleo))

4) Países donde se destacaron autores en los 60, cuyas pinturas fueron de un tamaño superior a los 100 cm2.
select : PAIS proyección:1
from : PAIS destaca AUTOR pintó PINTURA
where: (PAIS.tiempo[1] = 60 & PINTURA.tamaño[1] > 100)

6.2.2 Ahora presentamos otro esquema que pertenece a la información de una facultad:

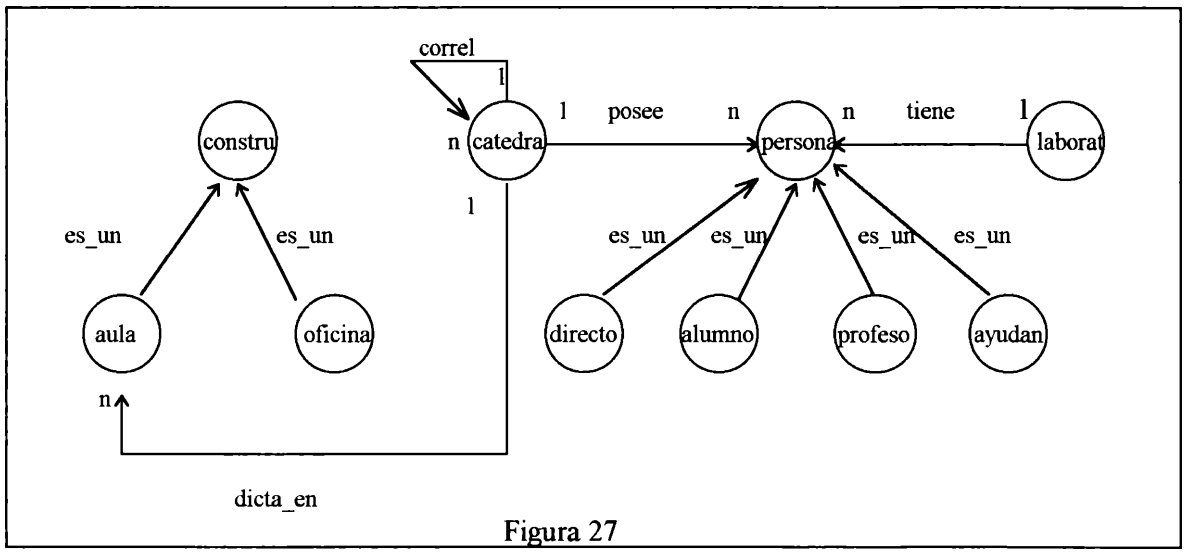


Figura 27

Este es un buen ejemplo porque muestra distintos conceptos que pueden estar involucrados en un esquema y pueden ser soportados por el servidor de consultas. Estos conceptos son:
1) Herencia: habiendo 2 posibilidades. Una de ellas es cuando existe un arco desde un tipo concreto a un subtipo concreto (en el ejemplo sería el arco "dicta_en", y significa que este arco aparecerá uniendo nodos del tipo CATEDRA con nodos del tipo AULA). La otra posibilidad aparece cuando tenemos un arco que une un nodo concreto con un nodo abstracto (en el ejemplo sería el arco "posee", este podrá unir entidades del tipo CATEDRA con cualquiera de los subtipos concretos de PERSONA (en este caso todos los subtipos son concretos, pero podría no serlo si se tuviera una factorización de clases más compleja (con más niveles)).
2) Recursión : es una herramienta muy potente a la hora de modelizar un esquema con estructuras de tipos que se repiten.

Ahora veamos los atributos de los tipos:
CATEDRA(nombre, carrera, duración, año,etc.)

PERSONA(nom_apell,dni,dirección,etc.)
PROFESOR(sueldo,..etc.)
ALUMNO(#alumno,..etc.)
AYUDANTE(sueldo, ..etc.)
DIRECTOR(sueldo,....etc)
CONSTRU (dimensiones, ubicación)
OFICINA(#escritorios)
AULA(#bancos)
LABORAT(nombre,..etc.)

Veamos las consultas que podríamos hacer:
1) Las correlativas inmediatas de Base de Datos
select : CATEDRA' proyección:1 CATEDRA' es un sinónimo de CATEDRA
from : CATEDRA correl CATEDRA'
where: (CATEDRA.nombre[1] = Base de Datos)

2) Alumnos que cursan POO y se encuentran en el LIFIA
select : ALUMNO proyección:1
from : CATEDRA posee ALUMNO + LABORATORIO tiene ALUMNO
where :(CATEDRA.nombre[1] = POO & LABORATORIO.nombre[2] = LIFIA)

3) Alumnos de Metodologías que son ayudantes de Programación.
La respuesta a esta consulta es la intersección de las 2 siguientes:

select : ALUMNO proyección:1
from : CATEDRA posee ALUMNO
where (CATEDRA.nombre[1] = Metodologías)

select : AYUDANTE proyección:1
from : CATEDRA posee AYUDANTE
where (CATEDRA.nombre[1] = Programación)

Esta intersección que se hace luego de las 2 consultas parecería en principio una intersección no muy bien definida, ya que se está haciendo entre conjuntos de distinto tipo: AYUDANTE y ALUMNO, cada uno de ellos con sus atributos. Esto es verdad, pero también es cierto que un ALUMNO es una PERSONA y que un AYUDANTE es también una PERSONA, por lo tanto la intersección se puede hacer en términos de comparar los atributos de PERSONA. A esta operación nueva la llamaremos PSEUDO_INTERSECCION, ya que no se comparan objetos del mismo tipo, sino objetos similares o hermanos (por pertenecer al mismo árbol de herencia). Esta operación actualmente no está definida en el servidor , pero se la propone como trabajo futuro , ya que la operación base de esta operación es la consulta, que sí se provee en el servidor.

6.2.3 El tercer ejemplo es muy parecido al ejemplo que se usó en el capítulo 2.

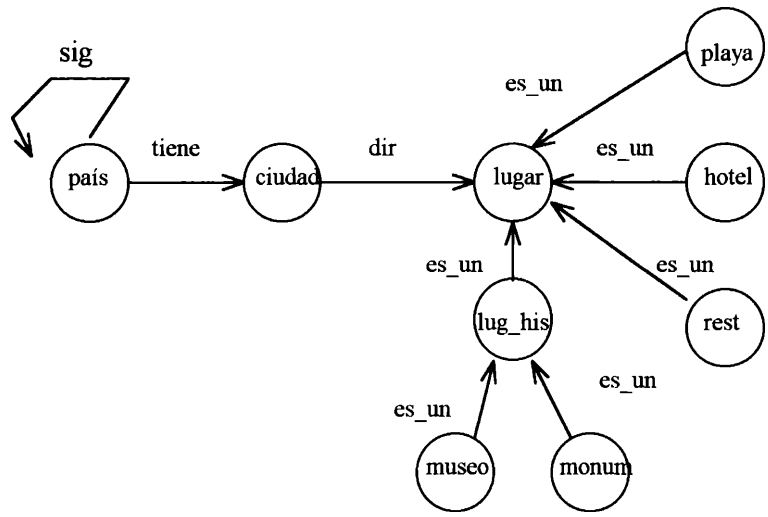


Figura 28

Como podemos apreciar este es un ejemplo rico en relaciones de herencia. A diferencia de los 2 ejemplos anteriores acá describiremos completamente como se le comunica este grafo al servidor.

Usaremos la sintaxis ToolBook (recordar que para aplicaciones ToolBook este código iría dentro de un handler llamado iniciar)

```
put sInitApp("agencia",9,3,153,7) into app_hand;
put GlobalLock(app_hand) into app_poi;
-- aca extraemos el código de error y la identificación de la aplicación - -
put pointerbyte(1,app_poi) into app_id;
put pointerbyte(0,app_poi) into error;

-- liberamos estructura --
get GlobalFree(app_hand);

if (error is 0) -- no hay error
  ------------------------------------------------------------------------
  -- primero comunicamos el esquema de la aplicación : tipos de nodos
  ------------------------------------------------------ tipos sinónimos
  ------------------------------------------------------ y tipos de links
  ------------------------------------------------------------------------
  -- agregamos tipos de nodos comunes                    99 : ascii del carácter 'c'
 get sAddTnode(app_id,"país",    "nombre sup",1,"si",99, null);
 get sAddTnode(app_id,"ciudad",  "nombre sup",1,"si",99, null);

 -- agregamos los tipos sinónimos
 get sAddSinon(app_id,"país","país1");
```

```
-- agregamos los tipos de nodos jerárquicos
get sAddTnode(app_id,"lugar",    "nombre",          1,"s", 114,null); -- 114 : 'r'
get sAddTnode(app_id,"lugar_his","año",             0,"i", 115,"lugar"); -- 115 : 's'
get sAddTnode(app_id,"museo",    "npisos",          0,"i", 115,"lugar_his");
get sAddTnode(app_id,"monum",    "altura material", 0,"is",115,"lugar_his");
get sAddTnode(app_id,"hotel",    "ncamas nest",     0,"ii",115,"lugar");
get sAddTnode(app_id,"rest",     "nmesas nest",     0,"ii",115,"lugar");
get sAddTnode(app_id,"playa",    "long",            0,"i", 115,"lugar");

--agregamos los tlinks
get sAddTLink(app_id,"sig","dist", 1,"país",  "país",  "i");
get sAddTLink(app_id,"dir","calle",1,"ciudad","lugar", "s");
get sAddTlink(app_id,"tiene","en",1,"país","ciudad","s");

- ------------------------------------------------------------------------
-- ahora agregamos los datos de la aplicación : nodos y links
------------------------------------------------------------------------
-- agregamos los nodos
get sAddNode(app_id,"país",  "Noruega 1500",    0);
get sAddNode(app_id,"ciudad","Oslo 25",         1);
-- notar que cuando agregamos un nodo de  tipo jerárquico , especificamos los valores
--  de los atributos comenzando por los atributos definidos en el tipo museo , luego
-- lugar_his  y luego lugar , es decir : desde la hoja hasta la raíz .
get sAddNode(app_id,"museo", "3 1850 Viking",   2);

 -- y así sucesivamente con los nodos , ahora agregamos los links
get sAddLink(app_id,"tiene","Noruega","país","Oslo","ciudad","sur",14);
  :
  :
end-if
end-if
```

Ahora haremos la siguiente consulta : queremos ver todos los pares de países tales que:
- estos países son adyacentes en el tour y distan no más de 300 Km. y
- el primer país tiene un restaurant de 4 estrellas o el segundo país tiene un hotel de al menos 90 camas

```
select : país sig país1                    proyección:2
from   : país tiene ciudad dir rest + país sig país1 + país1 tiene ciudad dir hotel
where : (sig.dist[2]  ≤300 & (rest.nest[1] = 4 | hotel.ncamas[3]>=90))
```

Hay 2 cosas para destacar en esta consulta : la primera de ellas es el uso de un tipo sinónimo, sin éste la consulta no sería posible de formularse y la segunda es que debido a la

estructura de la misma costaría mucho tiempo intentar recuperar toda esta información en forma manual si la AH superara unas docenas de nodos.

6.3 CLASIFICACION DE LAS APLICACIONES SEGUN SU GRADO DE ADAPTABILIDAD

En el capítulo anterior enunciamos cuales eran las cosas que había que agregarle a un aplicación para que pueda interactuar con el servidor. Luego de inspeccionar algunas aplicaciones y generadores de aplicaciones hipermediales nos damos cuenta que hay aplicaciones que se les puede agregar código para interactuar con el servidor y otras que (hoy en día) no. De aquí surge una clasificación de aplicaciones según su grado de adaptabilidad para integrarlas con el servidor (ordenadas de menor a mayor grado) :

1)"Aplicaciones completamente cerradas" : un ejemplo de este tipo de aplicaciones es el "Art Gallery", una enciclopedia en CD que no se proveen fuentes como para poder agregar funcionalidad para que interactúe con el servidor de consultas.

2) "Aplicaciones parcialmente cerradas" : este es el caso por ejemplo de un libro ToolBook, el cual una vez creado se le puede agregar código y/o manipularlo en forma dinámica, como para poder agregarle el código necesario para su interacción con el servidor. Incluso es de mucha utilidad poder definir módulos que se encarguen de proveer el comportamiento en común de un conjunto de aplicaciones generadas con un mismo generador, dejando solamente al autor de la aplicación la codificación de muy pocas líneas para completar la funcionalidad necesaria. Toolbook es un ejemplo de esto, se puede escribir un libro del sistema que se encargue de este comportamiento en común.

3) El mejor conjunto de aplicaciones según su grado de adaptabilidad no existe en la actualidad, ya que serían aquellas que se desarrollarían SABIENDO que ya existe un servidor de consultas el cual requiere que las aplicaciones que quieran conectarse al mismo puedan adaptarse a ciertas modificaciones. De esta manera el diseñador de esta aplicación o generador de aplicaciones desarrollaría todo en función a este CONOCIMIENTO ANTICIPADO y daría así toda la flexibilidad para que estas modificaciones sean hechas en forma simple.

En la sección 5.1 mencionamos 4 ítems relacionados a lo que hay que agregarle a una aplicación para que pueda interactuar con el servidor de consultas. La más compleja de ellas es la tercera: la existencia de un mecanismo para poder mostrar los nodos que forman parte del camino respuesta a una consulta. Esto incluye entre otras cosas poder deshabilitar botones y agregar una opción para volver al nodo inmediato anterior del camino. Si un diseñador conoce este requerimiento (entre otros), posiblemente podrá proveer un módulo para que dada una lista de pares <nodo,link> , se encargue de mostrar el primer nodo de esta lista , poniendo el foco de atención en el botón que corresponde al link que acompaña al nodo en el par y deshabilitar el resto de los botones del nodo. De esta forma el usuario podrá inspeccionar sólo los nodos que forman parte del camino y no irse por otros nodos que no forman parte del camino respuesta.

CAPITULO 7

TRABAJO FUTURO

CONTENIDOS

En este capítulo detallamos el trabajo que se plantea para el futuro. En la primer sección estudiamos la posibilidad de almacenar consultas. En la segunda sección especificamos una nueva forma de comunicar los datos y el esquema de una aplicación al servidor. La tercer sección está relacionada con las operaciones que se pueden hacer entre las aplicaciones que están conectadas al servidor. Por último, la cuarta sección está relacionada con el tratamiento de estructuras virtuales.

7.1 ALMACENAMIENTO DE CONSULTAS

Una vez que el servidor da la respuesta a una consulta (que esta respuesta será una lista con los caminos que satisfagan esa consulta), se le dará la posibilidad al usuario de almacenar esta respuesta en un archivo (.QRY) , para su posterior tratamiento. Este tratamiento podrá ser algunas de estas cosas:

a) El hecho de tener una consulta almacenada nos da la base para la implementación de links virtuales. Recuperando este archivo tendremos un conjunto de links definidos implícitamente por el mismo. Una forma de definir links podría ser la siguiente : cada link estaría definido entre el nodo que se encuentra en el principio del camino y el nodo que se encuentra al final del mismo. En la base de datos estos links no existen, de allí su denominación como links virtuales.

b) Guardando las consultas (y viéndolas como conjuntos de caminos), podemos realizar operaciones basadas en el Álgebra de Hiperwalks (como concatenación y otras) y operaciones basadas en la Teoría de conjuntos (como Unión , Intersección , etc.), dando la posibilidad de guardar el resultado de estas operaciones (también en archivos) para su posterior tratamiento.

El hecho de cargar una consulta presupone un overhead de trabajo para el sistema. Cuando se salva una consulta , hay un esquema y datos implícito sobre la cual se hizo esa consulta, que puede no coincidir con el esquema y datos que existe en la aplicación al momento de cargar esa consulta.

7.2 COMUNICACION DE LOS DATOS Y LA CONSULTA

Hay dos momentos donde existe comunicación de información por parte del usuario al servidor. La primera de ellas es cuando la aplicación comunica al servidor el esquema y los datos de la misma. El otro momento, que es posterior al anterior, es cuando el usuario desea realizar una consulta. Proponemos formas distintas de realizar lo descripto anteriormente.

7.2.1 COMUNICACION DE ESQUEMA Y DATOS

Hasta el momento la forma de crear y comunicar el esquema y datos de la aplicación es a través de funciones provistas por el servidor llamadas desde la aplicación.

Una alternativa sería crear una aplicación (aparte del servidor) que asista al usuario en esta tarea. Esta aplicación daría posibilidades para crear interactivamente el esquema e ir viéndolo en una ventana gráfica (donde el esquema se vería como un grafo).

También esta aplicación daría facilidades para comunicar los datos al servidor (que actualmente serían las llamadas a sAddNode/sAddLink). Para esto se estudiaría la posibilidad de absorber los valores de los atributos desde los nodos en sí (en caso de que los nodos tengan definidos atributos dentro del sistema de hipermedia que se usó para crearlos, por ejemplo en ToolBook : se tomarían los valores de los atributos desde las páginas). Pero aclaremos que siempre existiría la posibilidad de definir nuevos atributos para los nodos en el servidor , como se lo hace actualmente a través de un parámetro en la función sAddTnode/sAddTlink. El tema de absorber los atributos desde los nodos directamente sería una tarea especializada, ya que absorber atributos desde una página (en ToolBook) no es lo mismo que absorber atributos desde un nodo definido con otro sistema de hipermedia. Por

lo tanto, esta tarea debe ser escrita en el mismo lenguaje o desarrollada con el mismo sistema de hipermedia que se usó para crear la aplicación.
El resultado de esta aplicación sería un archivo .DAT que tendría el siguiente formato:

```
INIT
  sInitApp(..
TNODOS
  sAddTnode(..
     :
     :
TSINON
  sAddSinon(..
     :
     :
TLINKS
  sAddTlink(..
     :
     :
NODOS
  sAddNode(..
     :
     :
LINKS
  sAddLink(..
     :
```

La ventaja de este método es que se le ahorra una gran cantidad de trabajo al usuario. Por otro lado este método es más lento ya que el servidor leería el esquema y datos desde un archivo en disco y es claro que al servidor le va a llevar más tiempo leer este archivo y procesarlo que directamente recibir las llamadas de las funciones desde la aplicación , encontrándose ésta en memoria.

7.2.2 ESPECIFICACION DE LA CONSULTA

En la actualidad el usuario ingresa la consulta (es decir, los campos select, from, where) en forma textual. Una alternativa es proponer una interface gráfica. Para esto, se presentaría un grafo que representa el esquema de la aplicación y el usuario ingresaría la consulta usando este grafo. Dado que la cláusula from es una secuencia de walk_expressions, el usuario podrá ingresar cada walk_expression (we) en forma gráfica : encerrando con un rectángulo cada we que forme la cláusula from. Entonces tendremos una secuencia de rectángulos demarcados por el usuario (cada rectángulo encierra un we , que es un camino en el grafo). Cada rectángulo en la secuencia intersecta con el rectángulo predecesor y sucesor (ya que deben compartir al menos 1 tipo de nodo, para más detalle ver la sección 3.4).
Para ingresar la cláusula select tendríamos el mismo mecanismo, pero el rectángulo que usamos para encerrar el we que forma el select debería distinguirse del resto de los rectángulos (que son los que forman la cláusula from). Entonces, se podrá usar un rectángulo con otro color de línea y/o otro tipo de línea.

La cláusula where no se puede ingresar en forma gráfica, ya que no es un camino, sino que especifica condiciones que deben cumplir los caminos que satisfagan la cláusula from. Este sería el único campo que se ingresaría en forma textual.
Queda claro que esta forma (gráfica) de ingresar una consulta es más directa que la forma actual y reduce el trabajo del usuario. Por otro lado, esta forma no sería tan apropiada si tendríamos que graficar un esquema con gran cantidad de nodos y links, ya que contamos con un espacio limitado dentro de una ventana. Por lo tanto podemos concluir que esta forma es adecuada sólo cuando tratamos con esquemas pequeños.

7.3 OPERACIONES ENTRE APLICACIONES

Al estar en presencia de un servidor multitarea sabemos que el mismo tiene el esquema y datos de todas las aplicaciones que requieren de su servicio. Esto nos da la base como para definir operaciones entre aplicaciones , ya sea a nivel de esquema , datos e incluso a nivel de consultas efectuadas por cada aplicación.

7.4 TRATAMIENTO DE ESTRUCTURAS VIRTUALES

Las respuesta a una consulta define una **estructura** o **composición virtual** (una buena explicación de esto la tenemos en el capítulo 1 sección 3). Se podría agregar un tratamiento de estas estructuras, que incluiría la posibilidad de agregarles atributos. Estos atributos podrían ser nuevos (como por ejemplo agregar un atributo tiempo al camino devuelto en una consulta, que significaría el tiempo que llevaría recorrer ese camino) o podría ser un atributo derivado en función de los atributos presentes en cada entidad del camino (por ejemplo si tuviéramos una composición de entidades que poseen un atributo superficie, se podría tener en la composición un atributo derivado llamado también superficie, que contendría la superficie total ocupada por la composición, que sería una "suma" de las superficies de las entidades que componen la composición).

BIBLIOGRAFÍA

- [*AMMA*92] Bernd Amman , I.N.R.I.A. Miachel Scholl, Cedreic / CNAM. GRAM : a graph data model and query language. ECHT '92 Proceedings. Diciembre 1992

- [*CLUE*89] F. Bancilhon, S. Cluet y C. Delobel. A Query Language for the 02 Object-Oriented Database System. En DBPL'89, páginas 122-138, Salishan Lodge, Oregon, June 1989. Morgan Kaufmann.

- [*DAVI*93] Hugh C. Davis, Simon Knight and Wendy All. A light hipermedia link services: A study of third party application integration. ECHT '94, Septiembre 1994.

- [*HALA*87] Frank G. Halasz. Reflections on NoteCards: seven issues for the next generation of hipermedia systems. Hipertext '87 Proceedings.Noviembre 1987.

- [*HALA*90] Frank G. Halasz y M. Schwartz. The Dexter hypertext reference model. Proceedings of the NIST Hipertext Standarization Workshop. Enero 1990.

- [*HALL*93] Wendy Hall, Gary Hill, Hugh Davis. The Microcosm link service. Hypertext '93 Proceedings, Noviembre 1993.

- [*HALL*94] Gary Hill, Wendy Hall. Extending the Microcosm model to a distributed enviroment. ECHT '94, Septiembre 1994.

- [*HARA*71] F. Haray. Graph Theory. Addison Wesley Series in Mathematics, 1971.

- [*ICHI*93] Satoshi Ichimura, Yutaka Matsushita. Another dimension to hipermedia access. Hipertext'93 Proceedings, Noviembre 1993.

- [*MARM*92] Michael Marmann, Gunter Schalageter.Towards a better support for hipermedia structuring : The Hydesign Model. Acm Echt Conference, 1992.Diciembre 1992.

- [*MEND*89] M. Consens y A. Mendelzon. Expressing structural hypertext querys in GraphLog. Proccedings of Hipertext '89, Noviembre 1989.

- [*MEND*90] M. Consens y A. Mendelzon. GraphLog : a visual formalism for real life recursion. Proccedings of the ACM Symposium on Principles of Database Systems, Nashville, Tenessee, 1990.

- [*MEYR*89] Norman Meyrowitz.Hipertext - Does it reduce cholesterol, too ?.Basado en la conferencia Hipertext '89. Noviembre '89.

- [*NANA*91] Jocelyne Nanard, Marc Nanard Using structured types to incorporate knowledge in hipertext.. Hipertext '91 Proceedings. Diciembre 1991.

- [0293] 02 Technology. The 02 User's Manual, Version 4.4, Diciembre 1993.

- [*REBE*90] Rebeca J. Wirfs-Brok y Ralph E. Jhonson. Current research in Object Oriented Design.Communications of the ACM/Septiembre 1990/Volumen 33.

- [*RIZK*94] V. Christophides, I.N.R.I.A . A. Rizk, Euroclid. Querying structured documents with hipertext link using OODBMS. ECHT '94 Proceedings,Septiembre 1994.

- [*ROBE*81] Robertson G., McCracken D, Newwell A. The ZOG approach to man machine communication. Int. J. of Man-Machine studies, 14, 1981.

- [*SMIT*88] Jhon B. Smith, Stephen I. Weiss. Hipertext. Comunication on the ACM, July 1988.

- [*SCHW*91] Franca Garzotto, Paolo Paolini, Daniel Schwabe. HDM : A model for then design of hipertext applications Hipertext '91 Proceedings, Diciembre 1991.

- [*SCHW*94] Daniel Schwabe, Gustavo Rossi. 1994. OOHDM : An object oriented hipermedia design model.

- [*TAM*93] Nipon Charoenkitkarn, Jim Tam, Mark H. Chignell , Gene Golovchinsky.Browsing through querying: Designing for electronic books. Hipertext '93 Proceedings. Noviembre 1993.

- [YODE87] Robert Akscyn, Donald McCracken,Elise Yoder. KMS: a distributed Hipermedia System for Managing Knowledge in Organizations. Hipertext 87, Noviembre 1987.